

NO-A164 956

METHODOLOGY INVESTIGATION MULTILINGUAL STATIC ANALYSIS  
TOOL (MSAT)(U) ARMY ELECTRONIC PROVING GROUND FORT  
HUACHUCA AZ E L ANDERSON NOV 85

1/1

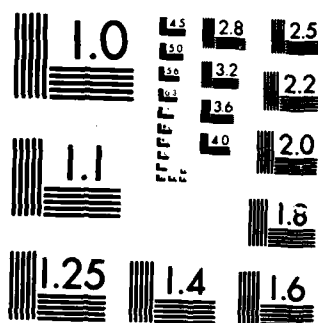
UNCLASSIFIED

F/G 9/2

NL

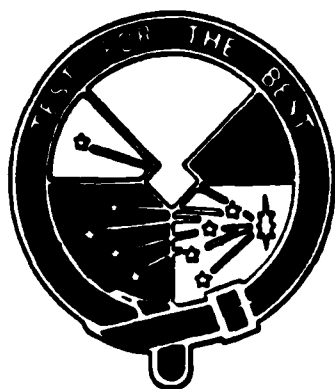

END

7/1/85



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

2



UNCLASSIFIED

DDC AD NUMBER \_\_\_\_\_

FUNDING PROJECT NO. IT665702D625

TECOM PROJECT (TRMS) NUMBER 7-CO-R85-EPO-007

TEST ACTIVITY REPORT NO. \_\_\_\_\_

TEST SPONSOR: U.S. ARMY TEST AND  
EVALUATION COMMAND

METHODOLOGY INVESTIGATION

FINAL REPORT

MULTILINGUAL STATIC ANALYSIS TOOL

(MSAT)

by

Edward L. Anderson

November 1985

AD-A164 956

US ARMY ELECTRONIC PROVING GROUND  
FORT HUACHUCA, ARIZONA

UNCLASSIFIED

DTIC  
ELECTE  
MAR 05 1986  
S E D

DTIC FILE COPY

### DISPOSITION INSTRUCTIONS

Destroy this report in accordance with appropriate regulations when no longer needed. Do not return it to the originator.

### DISCLAIMER

Information and data contained in this document are based on input available at the time of preparation. Because the results may be subject to change, this document should not be construed to represent the official position of the U.S. Army Materiel Command unless so stated.

The use of trade names in this report does not constitute an official indorsement or approval of the use of such commercial hardware or software. This report may not be cited for purposes of advertisement.



REPLY TO  
ATTENTION OF

DEPARTMENT OF THE ARMY  
HEADQUARTERS, U.S. ARMY TEST AND EVALUATION COMMAND  
ABERDEEN PROVING GROUND, MARYLAND 21005-8085

AMSTE-TC-M

27 JAN 1986

SUBJECT: Methodology Investigation Final Report, TECOM Project Number  
7-CO-R85-EP0-007

Commander  
U.S. Army Electronic Proving Ground  
ATTN: STEEP-MT-DA  
FORT HUACHUCA, AZ 85613-7110

1. Subject report is approved.

2. Test for the Best.

FOR THE COMMANDER:

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

*G. H. Shelton*  
GROVER H. SHELTON  
C, Meth Imprv Div  
Technology Directorate



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TRMS No. 7-CO-R85-EPO-007	2. GOVT ACCESSION NO. <b>AD-A164956</b>	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) METHODOLOGY INVESTIGATION FINAL REPORT-MULTI- LINGUAL STATIS ANALYSIS TOOL (MSAT)		5. TYPE OF REPORT & PERIOD COVERED Final Report
7. AUTHOR(s) Edward L. Anderson		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS US Army Electronic Proving Ground Fort Huachuca, AZ 85613-7110		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Test and Evaluation Command Attn: AMSTE-TC-M Aberdeen Proving Ground, MD 21005-5055		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS IT665702D625
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE Nov 1985
		13. NUMBER OF PAGES 84
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)  Software Measurement                      Software Static Analysis Tool Software Metrics                            Software Testing Software Quality                            Software Assessment		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Multilingual Static Analysis Tool (MSAT) investigation was conducted to develop a software tool to automate the collection and reporting of software design and quality characteristics in a multilingual milieu. The goal of MSAT is to minimize the manual effort associated with the static software assessment of a target software system's design, structure, maintainability, modifications and conformance with documented design and development standards. MSAT consists of a flexible, language-independent data collection component which extracts and stores items of interest in a DBMS; static analysis/report		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(UNCLASSIFIED)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

generation components for calculating and presenting software metrics and reports; and an executive control component which provides a user-friendly interface.

*... reflect measurement and*  
*... T...*

(UNCLASSIFIED)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## TABLE OF CONTENTS

	<u>Page</u>
FOREWORD.....	vii

<u>Paragraph Number</u>	<u>Page</u>
-----------------------------	-------------

### SECTION 1. SUMMARY

1.1 Background.....	1
1.2 Objective.....	2
1.3 Summary of Procedures.....	2
1.4 Summary of Results.....	2
1.5 Analysis.....	3
1.6 Conclusions.....	4
1.7 Recommendations.....	4

### SECTION 2. DETAILS OF INVESTIGATION

2.1 Software Test Methodology.....	5
2.2 Requirements Definition.....	5
2.3 MSAT Development.....	9
2.3.1 Concept of Operations.....	10
2.3.2 Design Goals and Approach.....	10
2.3.3 MSAT Requirements.....	12
2.3.4 Development Phases.....	15
2.4 MSAT Description.....	19
2.4.1 Overview.....	19
2.4.2 Detailed Design Aspects.....	26
2.5 MSAT Operational Aspects.....	32
2.5.1 Personnel Requirements.....	32
2.5.2 MSAT Operational Procedures.....	33
2.6 Future Development.....	35
2.6.1 Candidate Tasks for MSAT P <sup>3</sup> I.....	35
2.6.2 Software Test Methodology.....	37

Paragraph  
Number

Page

SECTION 3. APPENDIXES

A	Methodology Investigation Proposal.....	39
B	References.....	47
C	Acronyms and Abbreviations.....	51
D	Software Hierarchy Definitions.....	57
E	MSAT Glossary.....	63
F	Software Standards Comparison.....	73
G	Distribution.....	79

PREVIOUS PAGE  
IS BLANK

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1 MSAT Concept of Operations.....	11
2 MSAT Functional Components.....	13
3 MSAT CSCI Architecture.....	21
4 MSAT Data/Control Flow.....	22
5 MSAT Logical Data Base Files.....	25
6 MSAT Target Software System Information.....	27
7 Pre-Planned Product Improvement.....	36
8 Software Hierarchy.....	59

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
I Systems Requiring Assessment.....	6
II Language Processors Required.....	7
III Proposed MSAT Static Analysis Functions.....	16
IV MSAT Implementation Issues.....	17
V MSAT Initial Operational Capability.....	18
VI MSAT Development Phases and Products.....	20

PREVIOUS PAGE  
IS BLANK

FOREWARD

Ultrasystems Defense and Space Systems, Incorporated,  
Sierra Vista, Arizona assisted in the preparation of this document  
under Contract Number DAEA18-83-C-0003.

PREVIOUS PAGE  
IS BLANK

## 1. SUMMARY

### 1.1 Background

This document comprises the final report for the methodology investigation for the development of the Multilingual Static Analysis Tool (MSAT). The MSAT concept was conceived during earlier investigations on the Program Flow Analyzer (PFA). PFA research into software static analysis tools and associated software metrics culminated in the recommendation to develop a multilingual software analysis system [1].

The U.S. Army Electronic Proving Ground (USAEPG) and other Installation/Field Operating Activities (I/FOAs) have been tasked by the U.S. Army Test and Evaluation Command (TECOM) to perform software testing of systems containing embedded computer resources. Comprehensive software testing includes both the dynamic testing of performance and reliability (using instrumentation such as the Test Item Stimulator and Hybrid (hardware/software) Monitor) and the static testing of software quality (using tools like PFA and MSAT). Because complex system functionality in DoD systems is increasingly provided by software, the task of assessing performance and quality features of software is becoming a critical factor in the addition of viable systems to the inventory.

Static analysis tools, by examining the actual target system source code, provide visibility of the software design and quantitative measures of software quality as actually implemented. An obvious by-product is the ability to analyze the correctness of documentation with respect to the implementation. Less apparent are the support of dynamic analysis and various uses during Life-Cycle Software Engineering Center (LCSEC) activities (e.g., automatic documentation generation and version comparison). Use of static analysis techniques to support testing of maintainability issues is potentially highly cost-effective since up to 80 percent of software costs are associated with maintenance.

The test environment at USAEPG and other I/FOAs provides some unique challenges for software testing. Unlike many development/test organizations with a limited number of software languages and computer architectures, DoD systems are characterized by a plethora of languages and architectures. In addition to the multilingual requirement is the need to accommodate a number of evolving software development standards. On 10 systems tested at USAEPG, 13 software languages and 4 software standards were analyzed.

Justification for a multilingual/multistandard tool is obvious, given the unique environment. Automation of static analysis is difficult to dispute also, since, in one test alone, a cost savings of \$500,000 was realized with a static analysis tool relative to performing the same level of analysis manually. Besides being labor-intensive, manual static analysis tends to be error prone, less comprehensive, and far less consistent (because of "sampling" practices) than automated techniques. Further evidence of the utility of automated static analysis for software testing is provided in the references [2, 3, 4].

PREVIOUS PAGE  
IS BLANK

The MSAT investigation was conducted to develop a software tool to automate the collection and reporting of software design and quality characteristics in a multilingual environment. The goal of MSAT is to minimize the manual effort associated with the static software assessment of a target software system's design, structure, maintainability, modifications, and conformance with documented design and development standards. MSAT consists of a flexible, language-independent data collection component which extracts and stores items of interest in a data base management system (DBMS); static analysis (SA)/report generation (RG) components for calculating and presenting software metrics and reports; and an executive control component which provides a user-friendly interface.

## 1.2 Objective

The objective of this investigation was to develop MSAT, incorporating the proven concepts of previous investigations. Supplementary goals included the following:

- Reduction of the effort required to add languages and standards.
- Creation of a user-friendly man-machine interface.
- Incorporation of new metrics as identified by previous research and a design which facilitates augmentation of metrics, SA, and RG capabilities.
- Validation and configuration management of a single tool to replace the "family" of existing tools.

## 1.3 Summary of Procedures

Initial efforts on the MSAT investigation focused on determining the requirements for a static analysis tool to satisfy the stated objectives. Preliminary software requirements were derived by examining existing tools and test needs. These were developed into proposed specification documents and used as a basis for defining the needs of other I/FOAs. The preliminary specifications were then updated to reflect the initial operational capability (IOC) agreed upon by TECOM Software Technical Committee (TSOTEC) review.

Development of MSAT proceeded, following the guidelines established by the (then) proposed DOD-STD-SDS (now DOD-STD-2167). A Digital Equipment Corporation (DEC) VAX-11/VMS system with VAX FORTRAN, and the INGRES DBMS were provided as the host environment. VAX FORTRAN and Intel 8085 assembly were chosen as the first two target languages with DOD-STD-SDS as the default software standard. Plans for MSAT included provisions for training, sustaining, and configuration management of the tool. While part of the plan, these activities will not be initiated until completion and validation of the IOC.

## 1.4 Summary of Results

The MSAT investigation resulted in the development of a general purpose tool for software static analysis in a multilingual environment. Examination of projected test needs and a survey of existing tools revealed that a multilingual requirement, which was not satisfied by available software products, remained valid despite the trend toward Ada. MSAT software requirements were based on previous investigations at USAEPG, verified by other tool designs,

and coordinated with other TECOM I/FOAs. The investigation continued with the design and implementation of the MSAT IOC, following an incremental approach which stressed flexibility to allow for enhancements.

The MSAT design was centered around two reusable software components. An existing parser generator was used as the basis for a table-driven language processor to minimize the amount of language-dependent software in MSAT. INGRES, a commercial DBMS product, provided a flexible data base schema as well as data base management, user-friendly forms/menu features, and data retrieval mechanisms. Additional flexibility was designed into the system to allow adaptation to various software development standards and for supplementing the initial SA/RG capability.

Validation and configuration management of MSAT were planned as post-development activities. As such, these and the related activities of distribution, maintenance, and training remain to be accomplished.

One of the design goals of MSAT was to provide for future enhancements during the initial development. A number of candidate tasks were proposed for further improvement of the initial product. At a higher level, the need to maintain currency with advances in software test methodology was identified. A known deficiency in this area includes the lack of systematic methods for determining test coverage (thoroughness).

### 1.5 Analysis

The design and development approach of MSAT were obvious means of satisfying the stated objectives. First, the design was based upon previous efforts at USAEPG, with input from other I/FOAs and lessons learned by other tool developers. A table-driven technique provides the requisite reduction of effort to expand the language processing capability. Also, a commercial DBMS supplies extensible features and a user-friendly man-machine interface. Although numerous metrics, in addition to those implemented for IOC, are conceivable, provision for supplemental measures exists in the data base and SA/RG components.

Validation and maintenance activities associated with a software product such as MSAT are desirable after completion of the initial tool. A programmer/librarian function is normally the means applied to ensure that proper configuration management requirements are met. Additional support of users in applying MSAT to software assessment or LCSEC functions could be achieved by developing guidelines to assist the analyst. A by-product of these guidelines could be a standardized software assessment procedure for more consistent reporting and analysis.

Proposed enhancements to MSAT are best integrated incrementally in consonance with the evolutionary design philosophy. In accordance with this approach, investigations would be conducted to identify and prototype additional software quality metrics and SA/RG functions. After evaluation, desirable capabilities would be incorporated into a documented, production version of the tool. More advanced investigations are necessary in areas where little research has been accomplished or consensus is wanting relative to the validity of various metrics.

## 1.6 Conclusions

The MSAT development achieved the objectives and goals of the methodology investigation, contingent upon validation of the tool, training other I/FOA personnel in the use of the tool, and performance of sustaining obligations. MSAT currently provides an automated means to observe and measure various software quality features in an environment characterized by diverse language requirements. In this capacity, MSAT possesses unique capabilities not duplicated by available static analysis tools. Exploiting the inherent flexibility to enhance the initial capability was both pre-planned and essential to the viability of technologically current instrumentation.

## 1.7 Recommendations

The following recommendations, in order of precedence, are suggested for completely satisfying the objectives and fulfilling the needs determined during the investigation.

a. Validation of MSAT should be performed by an independent test activity. Following acceptance of the tool, sustaining functions including distribution, maintenance, training, and configuration management should be initiated. A desirable product from this effort is a set of guidelines for application of the MSAT.

b. An investigation should be conducted to supplement the basic features of MSAT by implementing the proposed enhancements. This would result in improved performance and additional static analysis capabilities. Priority should be assigned to those functions which would benefit the majority of users, both I/FOAs and LCSECs.

c. More advanced investigations should be initiated to examine advances in software test methodology. The objective would be to maintain currency with the state-of-the-art by developing quantifiable parameters to measure software attributes. Efforts should focus on deficiencies in current methodology with the goal of developing practical, as well as theoretically sound, solutions.

## 2. DETAILS OF INVESTIGATION

The MSAT investigation accomplished the development of a software static analysis tool suited to the multilingual test environment at the USAEPG. Software static analysis requirements were re-examined to identify current technology; requirements were formalized and reviewed for applicability by TECOM I/FOAs, and a fully documented tool was produced. The results of this effort are summarized below. Persons unfamiliar with the software hierarchy terminology endorsed by DOD-STD-2167 should refer to appendix D; terms associated with static analysis and MSAT may be found in appendix E.

### 2.1 Software Test Methodology

Software testing may be dichotomized into the complementary components of static and dynamic analysis. Static analysis tools and techniques examine program source code and software materials in a non-executable environment. Static methods inherently are able to analyze every software statement in a system, furnish visibility into design and quality, and provide a basis for dynamic analysis. In contrast, dynamic analysis examines characteristics of program performance. While analysis of every instruction may be impractical, dynamic tools can measure performance attributes which can only be collected during or after execution of the software. If the staggering cost of software maintenance is no object, dynamic analysis alone can demonstrate the balance of the critical issues: correct functionality, reliability, timing, and resource utilization. Comprehensive software testing exploits the synergism of both dynamic performance and static quality assessment.

### 2.2 Requirements Definition

Experience with previous static analysis tools had proven the utility and cost-effectiveness of automated techniques and demonstrated the feasibility of a language-independent tool. The experiential feedback from actual software testing served to identify those features of static analysis which contribute the greatest assistance to the software analyst. Although earlier investigations [1] identified the need for a multilingual tool and developed prototype tools based on the premise that such tools were nonexistent, these issues were re-examined for validity.

A compilation of software systems to be tested at USAEPG included more than 10 tactical systems, using twice as many languages. Table I lists a representative sample of the systems and the software languages. Viewed by level of language, table II, the fact that roughly half of the languages requiring processing are assembly (ASM) becomes readily apparent. These results are contrary to what would be expected from a review of software engineering principles and government standards which dictate the use of high order languages (HOLs).

Previous conclusions that a multilingual tool was required to satisfy test needs is clearly substantiated by the findings; nor has the need subsided in the interim. Although Ada may be expected to replace many of the current language requirements, other languages are gaining popularity for specialized applications (e.g., LISP for Artificial Intelligence). Use of very high order languages (VHOL) and specialized languages such as ATLAS (used in automated test equipment) is anticipated to continue in non-tactical systems. (The reason for EQUOL and VAX FORTRAN appearing as required languages will become obvious shortly.)

Table I.  
SYSTEMS REQUIRING ASSESSMENT (April 1984)

<u>SYSTEM UNDER TEST</u>	<u>LANGUAGE</u>
Teampack	ROLM 1602 ASM
RPV	FORTRAN IV (DEC) PL/M-80 SKC FORTRAN 8085 ASM MACRO-11 ASM SKC 3121 ASM
JTIDS	SKC FORTRAN SKC 3132 ASM AMZ 8002 ASM
REGENCY NET	MICROTEK PASCAL OMSI PASCAL 8085 ASM AMD 2901 ASM RCA 1802 ASM
TRAILBLAZER	C ROLM FORTRAN 68000 ASM

Table II.  
LANGUAGE PROCESSORS REQUIRED

<u>VHOL</u>	<u>HOL</u>	<u>ASM</u>
INGRES EQU*EL*	C	ROLM 1602
	FORTAN IV (DEC)	68000
	MICROTEK PASCAL	8085
	OMSI PASCAL	AMD 2901
	PL/M-80	AMZ 8002
	ROLM FORTRAN	MACRO-11
	SKC FORTRAN	RCA 1802
	VAX FORTRAN*	SKC 3121/3132

\* Implementation languages of MSAT.

Ancillary information on software development standards was acquired while compiling the language requirements for a static analysis tool. Frequently encountered were MIL-STD-1679, DOD-STD-1679A, and variations of these standards. DOD-STD-2167 (DOD-STD-SDS at the time) had not been approved but has now been promulgated and will supercede earlier development standards in future testing. Systems without a specified software development standard typically have the most recent standard applied during testing.

The impact of accommodating various software standards was assessed by a comparison of those most frequently encountered (see appendix F). This assessment indicated that some degree of flexibility was required, but enough commonality exists among the quantifiable criteria to assign standards flexibility minor importance. One aspect of the standards which did influence the design was the new terminology: unit vs module, computer software component (CSC) vs computer program component (CPC), etc. This presented difficulties beyond the initial expectations. Whereas it was easy to conceive of a tool which could label items either CSC or CPC, some terms were so well established in the literature that an abrupt change was hard to imagine (e.g., unit (module) coupling, internal routine (procedure)). A compromise was adopted whereby the new terminology would supplant the old except for well-established usage.

The multilingual requirements encompassing both HOL and ASM proved to be the most stringent criteria for selecting an existing tool to satisfy the needs. Ideally, if proponents of static analysis techniques were correct in assessing the utility of this method, numerous tools for performing static analysis functions should already exist. Indeed, they do. A survey of accessions in government and industry software library catalogs revealed hundreds of static analysis tools.

Though a list of tools and developers is too extensive for purposes of this report, a few sources were found to contain a majority of the available tools. The Federal Software Exchange Catalog [5], National Bureau of Standards (NBS) [6], and Software Research Associates [7] produce detailed descriptions of most of the static analysis tools of interest. (The NBS catalog has been acquired and revised by the Data and Analysis Center for Software (DACS), an Information Analysis Center of the Defense Technical Information Center (DTIC)). Before attempting to use these sources, one should be aware of the pitfalls in surveying test tools. One document which addresses the problems as well as provides descriptions of 43 tools was produced by the Software Test and Evaluation Project (STEP) [8]. It is sufficient at this point to realize that most of the existing tools are poorly documented, unavailable, lack flexibility and maintainability, and are limited functionally to a single target language.

While some tools address sophisticated testing issues, their single target language capability and host environment constraints preclude use at USAEPG. At the very least, a collection of tools would be required to satisfy the test needs--an untenable situation given the lack of documentation, incompatible design, diverse operating procedures, and challenges of conversion and maintenance. However, organizations requiring an analyzer for FORTRAN, COBOL, or JOVIAL could well find existing tools a viable approach to testing needs.

One approach to language independence in static analysis tools was the Automated Measurement Tool (AMT) development sponsored by Rome Air Development Center (RADC), and the U.S. Army Computer Systems Command [9]. Although the objective of the RADC work was to further the enhancement of software quality metrics and, as such, the AMT was experimental, it nevertheless possessed some general-purpose language processing features.

The AMT employed a syntax-directed (table-driven) parser with a language-dependent scanner to examine and automatically extract data items related to software quality. This information, in conjunction with manually entered data, was stored in a data base for analysis and report generation. With the exception of the LL(1) parser, the AMT was conceptually similar to language independent tools being developed at the USAEPG. Although the prototype AMT was more suited as a research instrument than a production tool, the similarity with USAEPG tools provided additional insight into the desirability of certain architectural features. Among the recommendations for further development of the AMT were:

- Addition of a form entry system.
- More flexible report generation services.
- Interface to a statistical package.
- Automation of the collection of additional metrics.
- Incorporate processing capability for another language.
- Expand data base capabilities.

Examination of the prototype USAEPG static analysis tools revealed shortcomings typical of other products. The family of PFA tools could be enhanced by modification of the following aspects:

- Reduction of effort for adding language capability.
- Addition of a user-friendly man-machine interface with forms/ menus.
- Creation of a single tool with similar operating procedures for all languages.
- New metrics and flexible means to accommodate new software standards and SA/RG functions.
- Completely documented and maintainable software.
- Validation of the tool and configuration management.

In summary, although existing tools provide advantages for particular applications, none provided a general purpose, flexible capability suited for a multilingual environment. An integrated tool bench constructed of individual component tools is not feasible, or at least not practical. Perhaps the greatest benefit to be derived from an examination of current offerings is the contribution to the design of a new tool.

### 2.3 MSAT Development

Once the requirement for multiple language capability was reconfirmed and a survey of existing tools offered design guidance, the effort to develop MSAT continued. Previously mentioned goals were retained and supplemented by lessons learned at the USAEPG and by other tool developers. A concept of operations (CONOPS) evolved at this time, followed by formalizing the software specifications and development plans.

### 2.3.1 Concept of Operations

MSAT was defined to be a software static analysis tool to automate the collection and reporting of target software design and quality characteristics. MSAT would provide software analysis data for development, in support of test and evaluation of systems under test (SUT) by I/FOAs, and for maintenance and software systems support by LCSECs. Figure 1 depicts the relationship among SUT software source, MSAT and MSAT reports, the analysis process, and the final product for test reporting. Use of MSAT for support activities is similar, though the final product is a modified software system with some documentation automatically generated by MSAT.

### 2.3.2 Design Goals and Approach

A user-friendly interface was conceived for MSAT which would provide the user software language, standards, analysis, and reporting options. Through an evolutionary development process that was preplanned, MSAT would grow in capability without premature obsolescence caused by changing language and standards requirements. Flexibility was perceived as a necessity to meet these goals.

Less critical, yet still important, was the desire to eliminate as many as possible of the shortcomings of previous tools and incorporate recommended modifications. Also important was the need to implement functionality with existing software in order to minimize development and maintenance costs.

Documentation was a key element absent from most existing static analysis tools. A software development plan was produced for MSAT which followed the guidelines of proposed DOD-STD-SDS. This served two purposes. First, it ensured adequate documentation, although the development plan did tailor the documentation required to a quantity commensurate with the size of the task. Secondly, DOD-STD-SDS (or its final form DOD-STD-2167) would be applied to future SUTs. Actual use of a standard for development would provide valuable experience for testing with the standard as criteria.

Consistent with this approach was the specification of software quality factors for MSAT which mirrored those used in software testing. One means of assessing conformance to the guidelines is through use of a static analysis tool. In this case, MSAT would be used as the tool to determine conformance to standards of MSAT software source code. In a sense, MSAT would be self-testing and automatically produce some of its final documentation (e.g., structure charts). This was the purpose for listing the MSAT implementation languages as a required capability.

Numerous functions are part of the domain of software static analysis. Because no single tool could include all the desirable functions in the IOC, a phased, or incremental, development was proposed for MSAT. This evolutionary approach possessed the following qualities:

- A subset of the final capabilities available prior to completion, allowing early feedback on desirable modifications.
- The IOC MSAT would contain limited SA functions and the capability for processing one HOL and one ASM language.

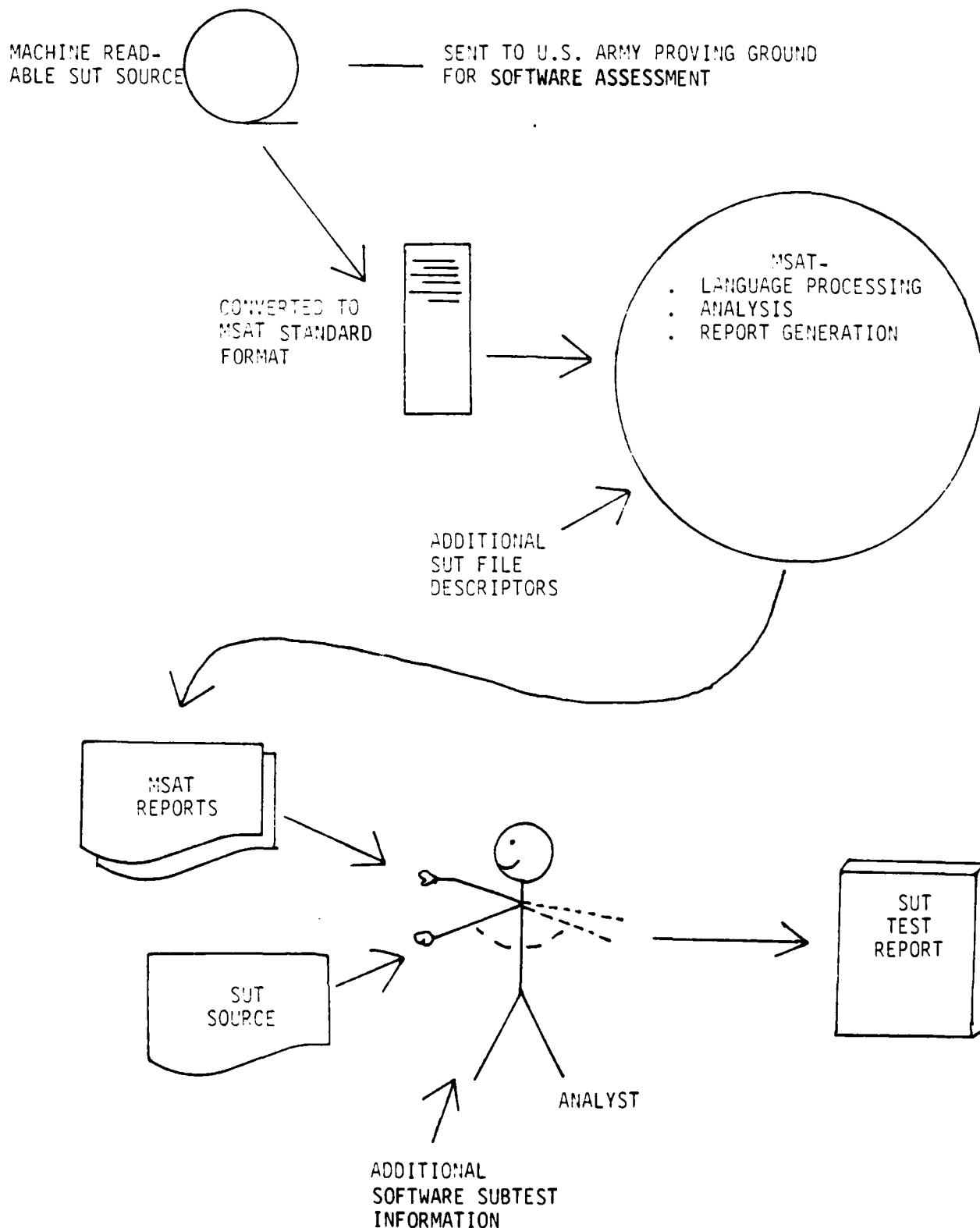


Figure 1. MSAT CONCEPT OF OPERATIONS

- A preplanned product improvement (P<sup>3</sup>I) approach to incorporate new functions and/or metrics not available initially, but deemed useful.
- A testbed for experimental language development and software quality metrics study.

Successful development of an incrementally produced product requires consideration of future capabilities and a design with clean functional separation of components and inherent flexibility. One of the hallmarks of such an approach is the definition of design items not essential to the immediate goal. Unlike a typical software development where extraneous software is considered a shortcoming, the early definition of functions and data items for P<sup>3</sup>I is to be encouraged. These objectives were applied first to formalization of MSAT requirements and sustained through the remainder of the development process.

### 2.3.3 MSAT Requirements

The description of the investigation (appendix A) included coordination with other I/FOAs in formulating requirements and reviewing the design of MSAT. Design reviews were scheduled to coincide with TSOTEC meetings up through preliminary design review (PDR). Coordinating requirements with other I/FOAs was more challenging since meetings were of insufficient length and too infrequent to allow for preparation of software specifications by committee. The solution was to draft preliminary specifications, based on past experience and results of the investigation, with review and comments by interested I/FOAs.

The major functional components of MSAT were easily defined since most mature static analysis tools employ conceptually similar architectures. Figure 2 illustrates the basic functions defined as follows.

- a. MSAT Executive Control (MEC). The purpose of the MEC is to provide a centralized component for the user interface, control the other MSAT components, and perform data base management.
- b. Automated Language Processing (ALP). The ALP scans the target source code to extract the data elements required for the various SA functions. Information is stored in the data base for further processing.
- c. Static Analysis. SA functions process the stored data elements to provide metrics or design information as desired. Output from SA is also stored in the data base.
- d. Report Generation. RG retrieves information from the data base to formulate output reports.

Having defined the basic architecture and major functions, additional requirements were proposed to accomplish the design goals mentioned above. The only remaining requirement of any significance, and one which demanded coordination with other I/FOAs, was determining which of the myriad of static analysis techniques should be implemented for the MSAT IOC.

The field of static analysis suffers from a lack of standardized terminology and dozens of overlapping categories. Obviously, the selection of MSAT SA functions would be quite difficult under these circumstances. An answer to

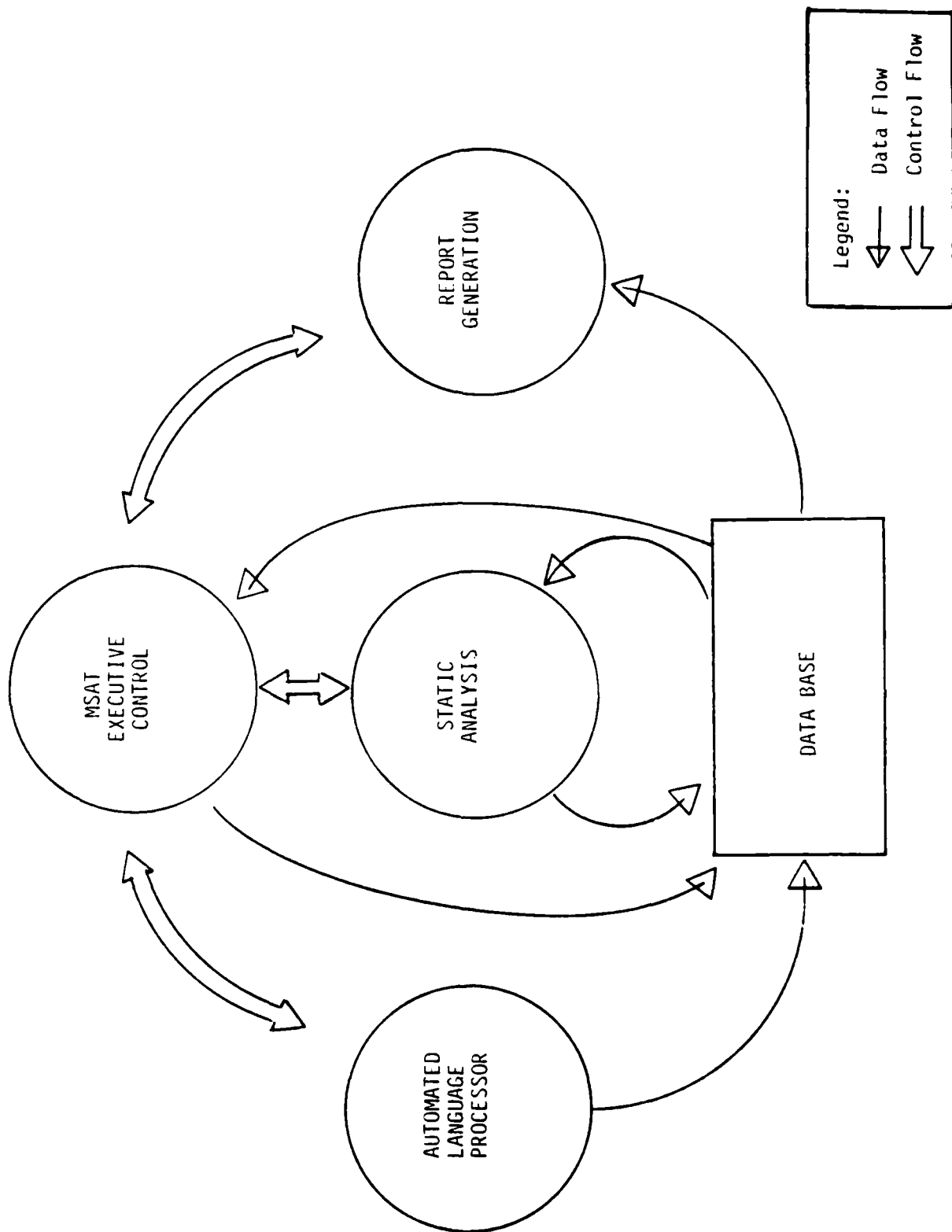


Figure 2. MSAT Functional Components

this problem was provided by an NBS publication on tool features for the Ada programming support environment [10]. The NBS report included a taxonomy of software tool features which included 15 entries under static analysis.

The contribution of the NBS taxonomy to selecting SA functions for MSAT was considerable. First, a manageable number (15) of categories was defined with a consistent terminology. Second, a prioritization of tool features was suggested, along with a discussion of various SA features. Additional utility was provided by the criteria used to select the listed functions. Only those features within the current state of software practice were listed (i.e., theoretical and experimental techniques not applicable to a production environment were excluded). Also of some consequence was the authoritative nature of the work, results having been reviewed by representatives from industry, government, and academia.

The static analysis functions derived from the NBS report were used as a generic list of potential SA functions in the MSAT System/Segment Specification (SSS). This was presented to the TSOTEC for prioritization and selection of initial MSAT capability. The following are the 15 functional categories.

- a. Auditing (standards compliance). Conducting an examination to determine whether or not predefined rules have been followed.
- b. Comparison (change analysis). Determining and assessing similarities between two or more items. In particular, performing change analysis on two versions of the same computer program to identify changes in the source code, documentation, or hierarchical structure.
- c. Completeness checking. Assessing whether or not an entity has all its parts present and if those parts are fully developed. A tool that examines the source code for missing parameter values has this feature.
- d. Complexity measurement. A method of determining how complicated an entity is (e.g., module . . . system) by evaluating some number of associated characteristics.
- e. Consistency checking. The determination of whether or not an entity is internally consistent in the sense that it contains uniform notation and terminology, or is consistent with its specification. For example, checking for consistent usage of variable names or consistency between design specifications and code.
- f. Cross-reference. Referencing entities to other entities by logical means. In particular, a cross-reference could illustrate all the variables and routines referenced by a unit.
- g. Data flow analysis. A graphical analysis of the sequential patterns of definitions and references of data.
- h. Error checking. The determination of discrepancies, their importance, and/or their cause (e.g., identification of possible program errors, such as misspelled variable names, arrays out of bounds, and modifications of a loop index).

i. Interface analysis. The checking of the interfaces between program elements for consistency and adherence to predefined rules and/or axioms. In particular, checking parameter usage (type, number) in calling and called routines. Determining the various degrees of module coupling might also be included in interface analysis.

j. Input/output (I/O) specification analysis. The analysis of the I/O specifications in a program, usually for the generation of test data.

k. Scanning. Examination of an entity sequentially to identify key areas or structure. For example, examining source code and extracting key information for generating documentation or source analysis.

l. Statistical profiling (analysis). Performing statistical data collection and analysis on software source code.

m. Structure checking. Detecting structural flaws within a program (e.g., recursive calls, calls to a top-level routine, reference to undefined routines).

n. Type analysis. The evaluation of whether or not the domain of values attributed to an entity are properly and consistently defined.

o. Units analysis. The determination of whether or not the units or physical dimensions attributed to an entity are properly defined and consistently used, ensuring variables used in computations have proper units (e.g., hertz--cycles/seconds).

Table III indicates the five functional categories agreed upon by the TSOTEC for IOC. In addition, structure checking was required by the USAEPG to retain functionality provided by existing tools. Furthermore, a minimal amount of error checking would result as a by-product of other SA functions. Selection of these categories was partially based on the usefulness demonstrated during software testing. Additional emphasis was placed on not duplicating functions provided by support software (e.g., cross-reference).

Subsequent reviews of the MSAT development resulted in the definition of implementation issues. The host environment for the initial version was specified as a VAX-11/VMS architecture with VAX FORTRAN as the primary language and the INGRES DBMS. The arguments for and against these choices are listed in table IV.

The preliminary set of requirements was expanded to produce the MSAT Software Requirements Specification (SRS). This document provided detail to further refine the specifications (since SA functions are never comprehensive for a given category) and develop the software design. Although further requirements surfaced throughout the development (e.g., enhanced security controls, automated source instrumentation, embedded procedure and language capability), the basic requirements for IOC remained as summarized in table V.

#### 2.3.4 Development Phases

The specification documentation, SSS and SRS, were produced during the requirements definition phase. These specifications are analogous to the system (or A-level) specifications of previous development standards. Because

Table III.  
PROPOSED MSAT STATIC ANALYSIS FUNCTIONS

Initial Implementation	1. Auditing
	2. Complexity Measurement
	3. Statistical Analysis
	4. Interface Analysis
	5. Comparison
	6. Consistency Checking
	*7. Error Checking
	**8. Structure Checking
	9. Completeness
	10. Data Flow Analysis
	11. I/O Specification
	12. Cross-Reference
	13. Scanning
	14. Type Analysis
	15. Unit Analysis

\* Produced as a byproduct of other functions

\*\* Required to retain current tool capability

Table IV.  
MSAT IMPLEMENTATION ISSUES

VAX-11 Implementation

- + Available for MSAT Development (USAEPG, Ultrasystems)
- + Standard Architecture for LCSECs
- + Ada Language System (ALS) available for P<sup>3</sup>I
- Availability at other I/FOAs

VAX FORTRAN

- + FORTRAN dialects widely used
- + Interfaces to DBMS
- + "Better" than F-77
- Data structures not suited to string, list processing
- Not as portable as Ada will be

INGRES DBMS

- + Increases flexibility
- Reduces development effort
  - Time
  - Risk
  - Cost
- + Government policy to utilize existing software packages (SDS)
- + Provides:
  - File management
  - Forms management
  - Report Writer
  - Graphics
  - Statistics
- + Isolates data base functions
- May limit portability

Table V.  
MSAT Initial Operational Capability

<u>Languages</u>	<u>SA Functions</u>	<u>Reports</u>
VAX FQPTRAN	Auditing	Standards Compliance
8085 ASM	Complexity Measurement	Source Listing/Table of Contents
(INGRES EQUOL)	Comparison	Change Analysis
	Interface Analysis	Interface Analysis
	Statistical Analysis	Software Quality Metrics
	Structure Checking (Error Checking)	Structure Chart (Error Reports)

p3I For:

Other SA Functions/Metrics, Library  
of Target System Software Languages

( ) - Minimal capability for IOC

DOD-STD-2167 introduces new terms for software products, table VI, listing the development phases and products, is supplied. The contents of other documents will not be described further in this report since the MSAT documentation is available upon request from the USAEPG.

## 2.4 MSAT Description

The following section provides a brief description of MSAT. Documentation listed in table VI should be referred to for a comprehensive understanding of the design and operational aspects.

### 2.4.1 Overview

Static analyzers are generally composed of four major components: language processor, data base, error analyzer, and report generator [8]. MSAT follows this basic architecture, augmented by features to provide a multilingual and flexible capability. User interface and control functions are isolated, along with multiple language/standards/terminology and data base management functions, in the executive control component. Language processing and the data base have design features tailored to the multilingual requirement; the error analysis/report generator functions are modularized in the SA/RG components to readily allow expansion. Figure 2 shows these major components, described further in the following paragraphs.

MSAT was designed as a single computer software configuration item (CSCI) comprised of CSCs and units. The executive CSC, called the MEC, controls the environment of other CSCs. The major components defined earlier were assigned the status of top-level CSCs (TLCSCs) and functionally decomposed into other TLCSCs. Figure 3 shows this architecture, except for the individual SA and PG functions, and the hierarchical identification nomenclature assigned each function.

Overall data and control flow is depicted in figure 4. The input and output data flows shown are described below:

- a. Annotated Source. The input source which has been tagged with key symbols by the ALP for use by the SA/RG functions.
- b. Default Standards. Predefined standards (e.g., MIL-STD-1679A) which may be used for determining standard's compliance in lieu of or in addition to user-defined standards.
- c. External References. A list of references supplied by the user which are external to the target software source which would otherwise be identified by MSAT as undefined references (e.g., operating system or library routines).
- d. Manual Data Entry. Data which must be entered manually into the data base. For example, data which cannot be collected automatically or data collected which must be supplemented or modified prior to analysis.
- e. Metrics. Software metrics produced by the SA function; includes summaries, statistics, counts, etc.

Table VI.  
MSAT DEVELOPMENT PHASES AND PRODUCTS

<u>Phases</u>	<u>Inputs</u>	<u>Products</u>	<u>Milestones</u>
Define (Requirement)	SSS	Software Requirements Specification (SRS)	Software Requirements Review (SRR)
Design (Preliminary)	SRS	Software Development Plan (SDP) Software Top-Level Design Document (STLDD) Data Base Design Document (DBDD) User's Manual (Preliminary menus/reports)	Preliminary Design Review (PDR)
Detailed Design	STLDD	Software Development Folders (SDF) Software Test Plan (STP)	Critical Design Review (CDR)
Construct	SDFs	Software User's Manual (Updated) Source, Object, Executable Files Software Product Specification (STLDD, DBDD, SDF) Software User's Manual (SUM) Software Programmer's (Maintenance) Manual (SPM)	
Test	Source, Object, Executable	Test Results  Version Description Document	End Product Acceptance (FQT)

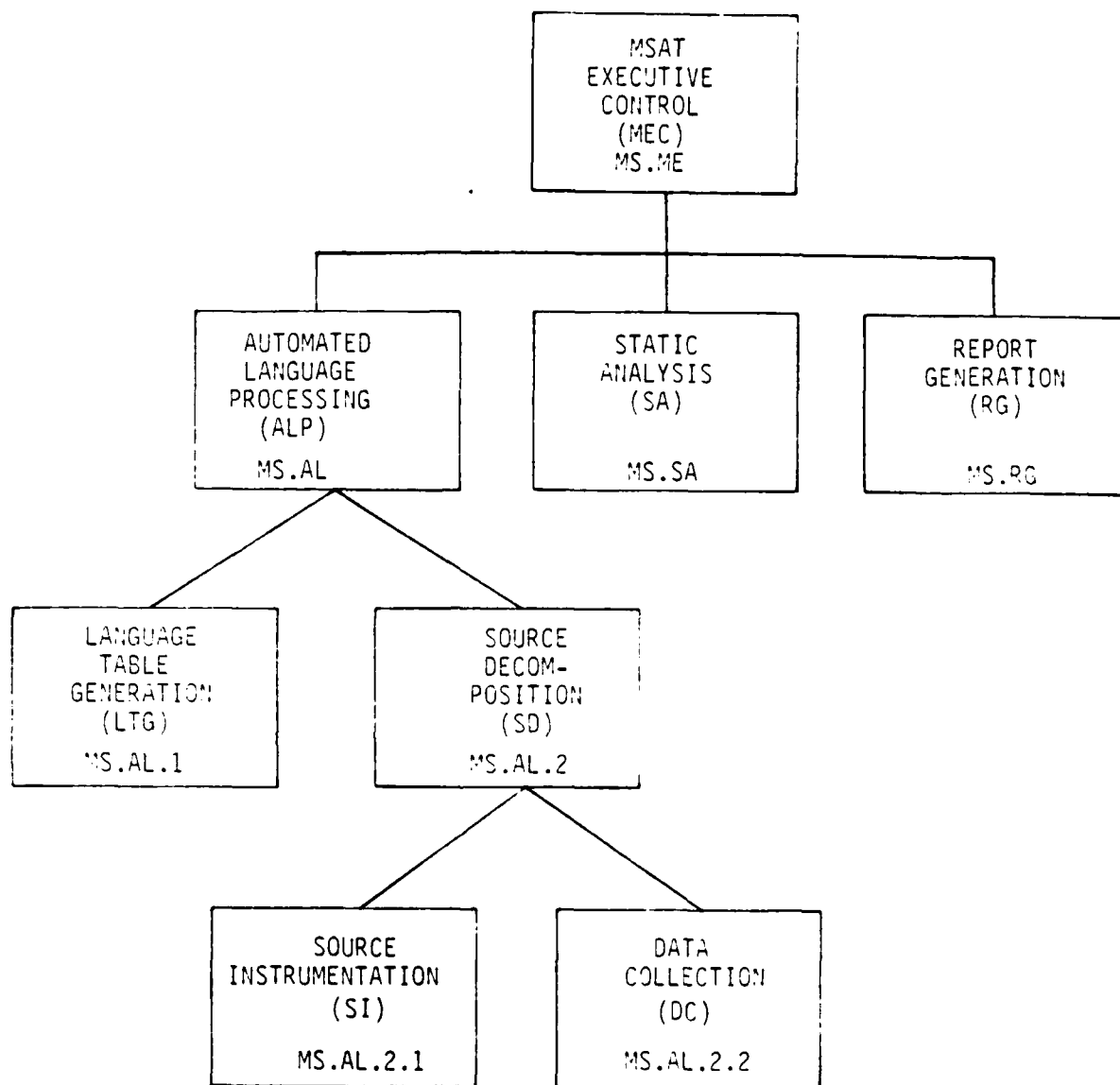


Figure 3. MSAT CSCI Architecture

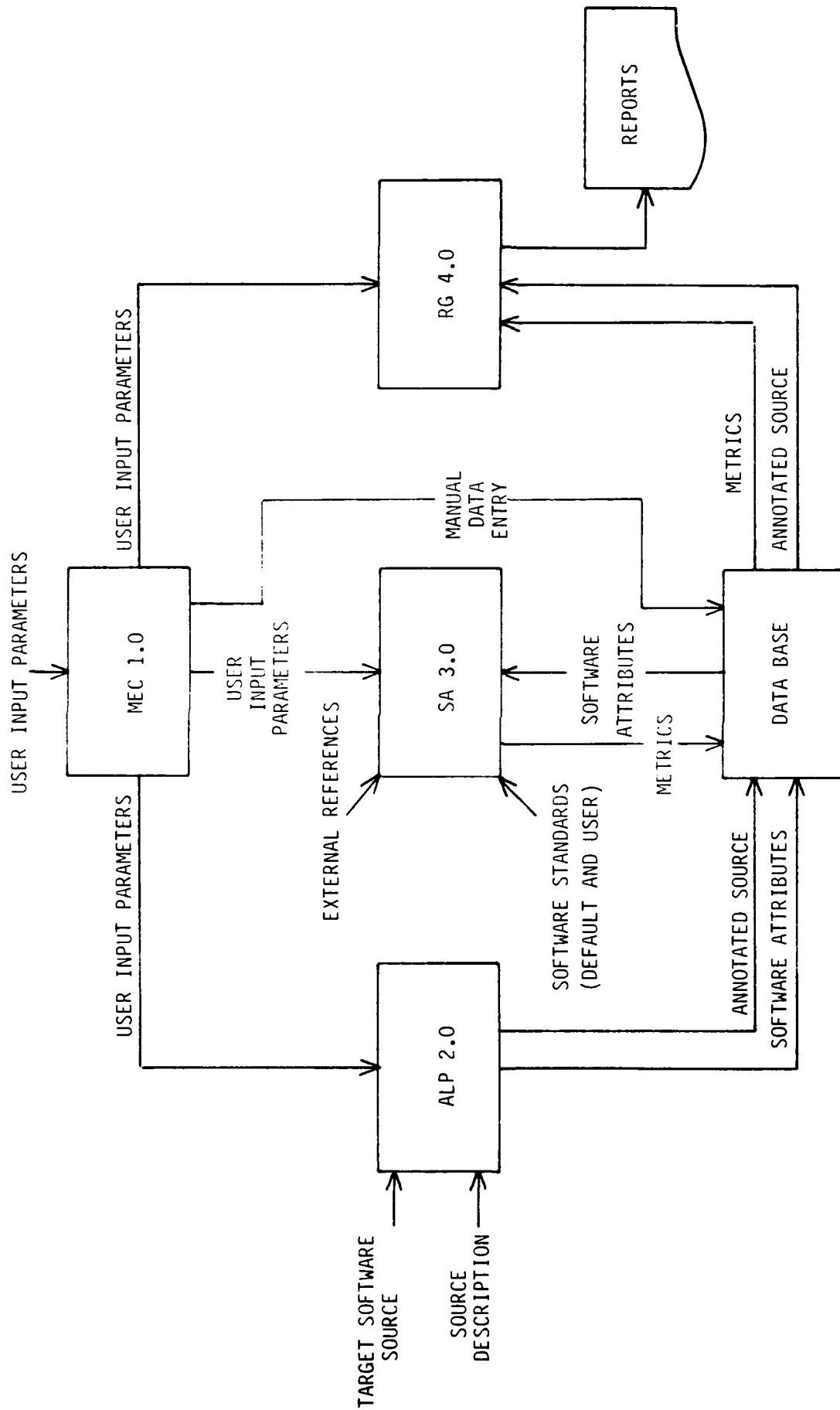


Figure 4. MSAT Data/Control Flow

- f. Reports. Reports output by the RG function.
- g. Target Software System (TSS) Source. The original (raw) source code file which will be transformed into the MSAT standard input format. The source code may be composed of HOL with embedded ASM, embedded VHOL, or both.
- h. Source (Language) Description. The language grammars (i.e., modified Backus-Naur Form (BNF) for HOL and ASM).
- i. Software Attributes. Those attributes of the software which are extracted by the ALP (e.g., number of lines of code (LOC) per unit, control structure, etc.).
- j. User Input Parameters. The data and control information which will be input by the user through the MSAT user interface.
- k. User Standards. Standards (criteria values) input by the user to be used for standards compliance, e.g., SUT-specific standards. A user might also specify a specific set of predefined standards which will be retained for reference in the MSAT data base: MIL-STD-SDS, MIL-STD-1679A, etc.

#### 2.4.1.1 MSAT Executive Control

The MEC function is the user's interface to the MSAT system. Inputs are via a forms/menu facility provided by the INGRES DBMS. MEC performs the following major functions:

- o User interface through forms/menus with input validation/recovery and on-line assistance.
- o Data base management: initialization, data entry, and language/standards library maintenance.
- o Initiation and control of the ALP, SA, and RG interactively or as batch processes.

#### 2.4.1.2 Automated Language Processing

The ALP performs functions related to automated language processing of TSS source code. The ALP includes language table generation (LTG) and source decomposition (SD) functions. (SD consists of source instrumentation (SI) and data collection (DC) functions.) These generate source language descriptions and extract the TSS information (TSSI) from MSAT standard input files (MSIFs) for storage in the MSAT data base (MSDB).

##### 2.4.1.2.1 Language Table Generation

The LTG is an independent process which creates the MSAT language-processing capability for each target programming language. The LTG requires language descriptions to generate the language-specific tables which are used by the SD function to drive an LR(1) parser to recognize specific constructs when that language is scanned.

#### 2.4.1.2.2 Source Instrumentation

The SI process performs an initial scan of the TSS source code to insert (where possible) instrumentation lines which identify items such as the beginning and ending of units, beginning and ending of internal procedures, and language context switches. SI consists of a number of instrumenters for converting raw source code to an MSIF, adding instrumentation lines, and preprocessing source code for the subsequent parser processing. Instrumenters are tailored to a specific language or class of languages.

#### 2.4.1.2.3 Data Collection

The DC process consists of a scan of the MSIF to parse source code in individual units and collect the data items required for the SA and/or RG functions. These collected items are placed in the MSDB for further processing or reporting. The user may specify that the entire MSIF or a selected subset of the units be parsed for data collection.

#### 2.4.1.3 Static Analysis

The SA component of MSAT uses the TSSI collected by the ALP and supplemented by manually entered data in the MSDB to perform various static analyses. Results are returned to the MSDB for reporting by RG.

#### 2.4.1.4 Report Generation

The RG function retrieves TSSI from the MSDB at the specified hierarchical level (or cluster of units), calculates any metrics dependent on the grouping, and generates reports (files or hard copy).

#### 2.4.1.5 MSAT Data Base

The MSDB provides storage and retrieval of information related to the target software source and MSAT system. Four major categories of information are defined: MSDB information, source language descriptions, software standards (criteria), and TSSI.

The major data base files, corresponding to the four major categories of information, are depicted in figure 5. The purpose of each file (implemented as a set of INGRES tables) is described below:

a. MSDB Information. The MSDB information contains MSAT system information and information on the location of other data. Access to all information in the data base (both INGRES tables and VAX files) is initiated through reference to these tables.

b. Source Language Description. The source language description consists of a library of grammar rules, syntax tables, and source preprocessors.

c. Software Standards. The software standards information is capable of containing a library of default standards (e.g., 1679A, SDS) and user-defined criteria.

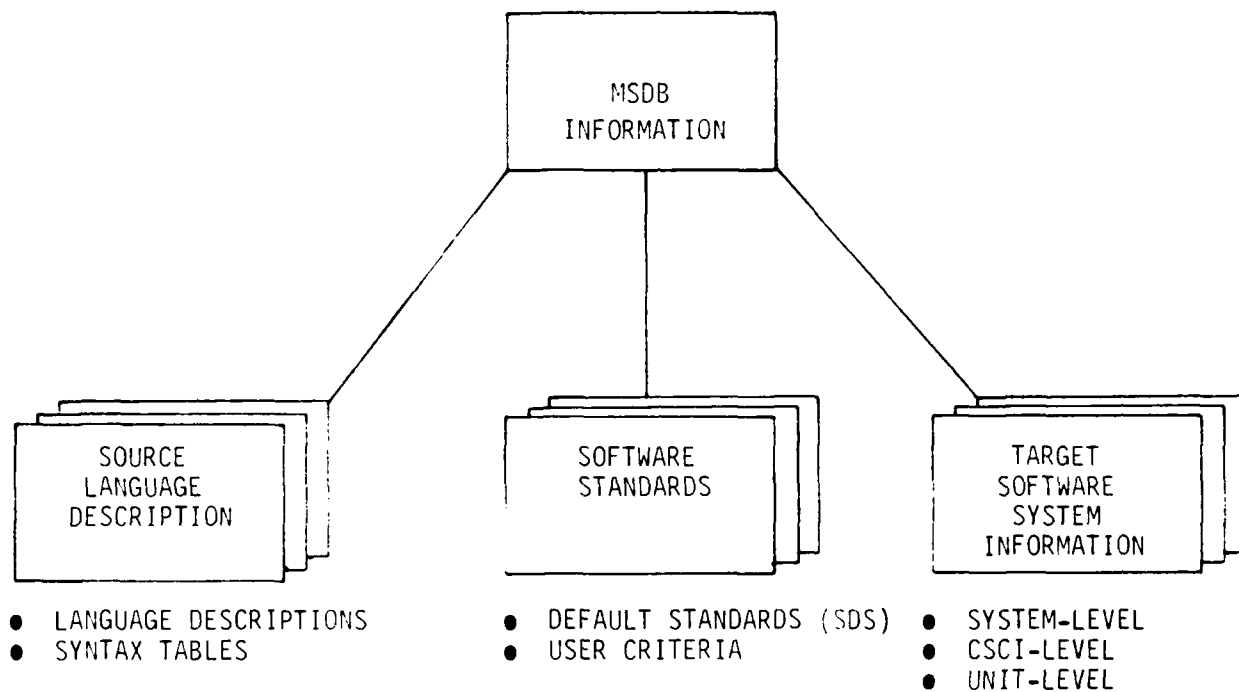


Figure 5. MSAT Logical Data Base Files

d. TSSI. The TSSI data consists of information at the system, CSCI, and unit levels for the TSS. The TSSI contains information from the implementation (code) phase of the software, although the flexibility exists for future inclusion of information from the design and test phases. Standards and environment data (e.g., external reference definitions) are capable of being specified at the CSCI level. Figure 6 shows the logical organization of data within the TSSI.

#### 2.4.2 Detailed Design Aspects

The following paragraphs expand on the overview above to provide some additional detail of the MSAT design. Again, the reader is referred to MSAT documentation for a comprehensive understanding since only the salient features are described herein.

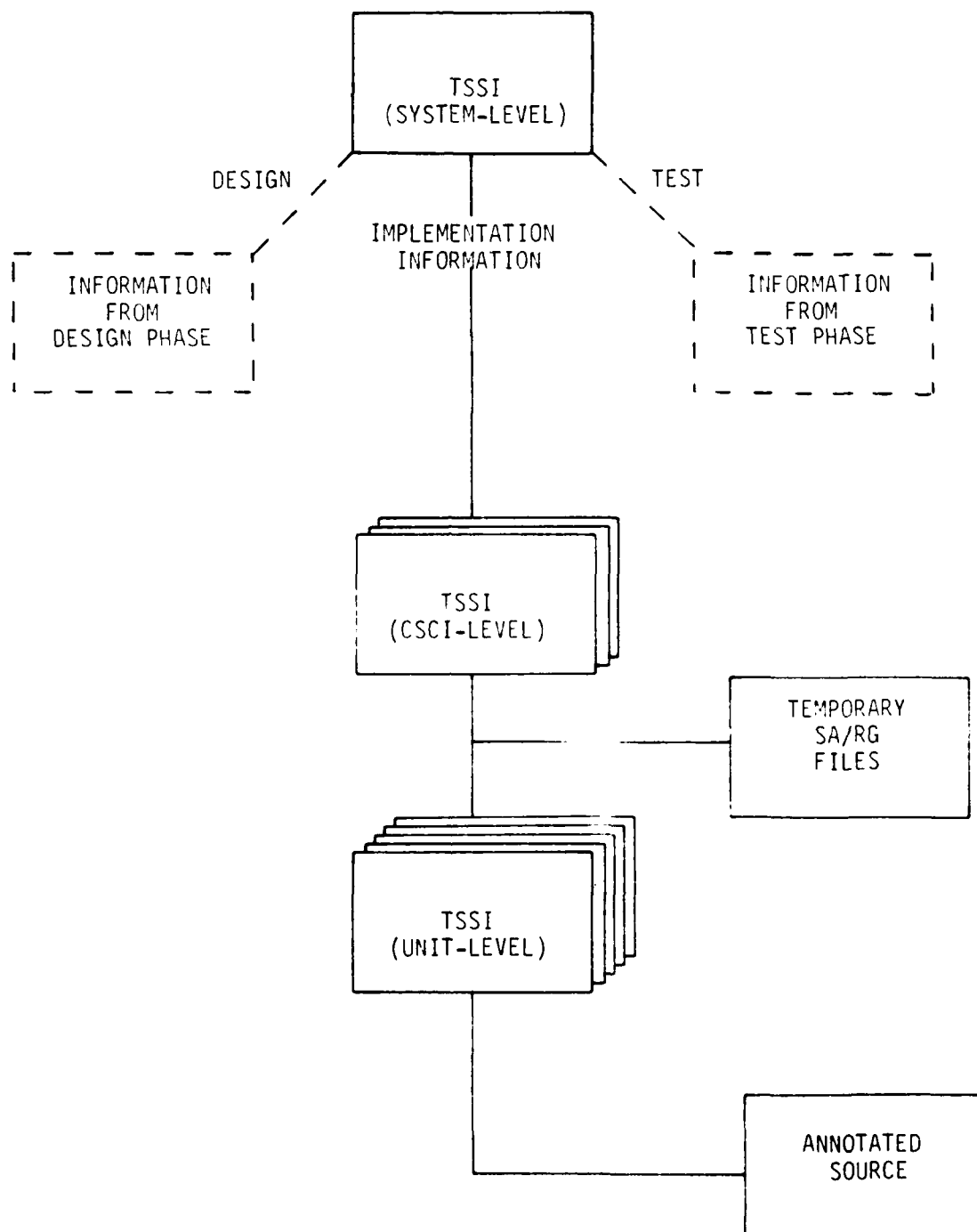
##### 2.4.2.1 MSAT Executive Control

MEC is the first component executed upon initiation of the MSAT system. Six functional operations are entered through the top-level menu:

- a. TSSI - Invokes the TSSI menu/form to allow:
  - Initial MSDB initialization for a TSS.
  - CSCI data entry.
  - CSCI language descriptions.
  - TSS standards specification.
- b. Language Installation - Invokes the LTG menu/form to allow the creation of, or MSDB insertion of, new source language description tables.
- c. Process MSIF (TSS Source Code) - Invokes the SD menu/form to allow:
  - Automated MSIF instrumentation.
  - Data collection.
  - MSIF editing.
- d. SA - Invokes the SA menu/form to allow the user to specify SA functions for a TSS or groups of units (clusters).
- e. RG - Invokes the RG menu/form to allow a user to specify various reports for a TSS or groups of units (clusters).
- f. Manual Query/Entry - Invokes the Manual Query/Data Entry menu/form to allow more flexible queries or specific manual data entry by a user.

Forms/menus are provided within these major functional categories to obtain additional levels of detailed data and control information.

The various processes within MSAT may be initiated in an interactive or batch mode. In the interactive mode, the user waits for the requested action (menu selection) to complete, and the results are displayed on the terminal and written in a user log file. In the batch mode, the requested action or group of actions is submitted as a batch process to the VAX/VMS system and all



---P<sup>3</sup>I CAPABILITY

Figure 6. MSAT Target Software System Information

output is directed to the standard output device and the user log file. When a user specifies a menu selection (e.g., the default SA functions) in the batch mode, appropriate commands are written in an MSAT batch command file. Each requested batch action is placed in this file in the order requested. This batch command file is normally submitted when the user "exits" MSAT (or optionally by another menu command).

A trace mode, available in a brief, verbose, and test form, provides the capability to generate additional details about the execution of the various processes within MSAT (i.e., LTG, SI, DC, SA, and RG). Use of the trace mode assists in the following types of activities:

- Verifying the correct placement of instrumentation lines within the MSIF (DC).
- "Debugging" new language tables (LTG).
- Determining the source of error in inaccurate report data (RG).

MSAT will accommodate three types of users: normal users who would run MSAT to process TSS source code and produce reports, advanced users who would add language and standards capability, and super users for data base administrator functions. All levels of users are assisted in performing their activity by help menu options.

#### 2.4.2.2 Automated Language Processing

The ALP provides the capability to automatically scan the MSIF, identifying and collecting the data elements required to perform the various SA functions and to generate the required reports. The ALP is made up of two major CSC divisions: LTG and SD. LTG generates the language-specific tables required for the SD to identify and collect the necessary data from the MSIF.

The SD scans the MSIF, instrumenting the TSS source and/or collecting the data elements required for the various SA functions. The SD is logically broken into the following subfunctions: SI and DC. The LC function uses the language parsing tables (which are produced by the LTG and installed in the MSDB) to drive the LR(1) parser to recognize language-specific constructs. The SI attempts to partially instrument and otherwise prepare the MSIF for processing by DC. The DC gathers data on each of the TSS units passed to the SD, and stores the data collected in the data base.

The ALP functions, LTG and SD (SI and DC), are described in the following paragraphs.

##### 2.4.2.2.1 Language Table Generation

The purpose of the LTG function is to generate a set of tables used for recognizing and processing source code of a given language. These tables must exist before any source code of the language can be processed by other MSAT functions.

The LTG generates parse, semantic, and token tables for a given source language from input descriptions of the language (augmented BNF) and its terminal symbols (regular expressions). Together, these tables allow the recognition of language constructs and the collection of construct-specific data. In this way, automatic processing of the source code occurs according to the user-specified language definition.

The LTG function requires a working knowledge of compiler tool usage, namely the LR(1) parse table generator and a scanner generator. These tools require the usual inputs of an augmented BNF grammar and regular expression file, respectively. Semantic action routines are specified in the augmented BNF by mnemonics.

There are two major algorithms within the LTG function: the LR(1) parser generator algorithm and the token table generator algorithm.

The LR(1) parser generator, a tool called LR, was originally developed by the Lawrence Livermore Laboratories [11]. LR is one of the existing tools (the INGRES DBMS being another) used to reduce the risk and lower the development effort of MSAT. (LR is listed in the Federal Software Exchange Catalog [5].) The algorithm, as implemented in MSAT, is based on an article appearing in *Acta Informatica* [12]. This algorithm creates LR(1) parse tables from an input BNF grammar description of a target software language.

The first step in the algorithm is to read the BNF grammar, find all productions, terminal and nonterminal symbols, and detect any syntactic errors in the BNF grammar statements. The next step is to find the goal symbol of the grammar. The goal symbol is the nonterminal symbol which all statements eventually reduce to, and is assumed to be the first nonterminal symbol in the BNF. The grammar symbols are then sorted so that all terminals precede nonterminals and both sublists are alphabetized. Two verification checks are performed: checking to see that all symbols are connected to the goal symbol, and checking to see that no nonterminals are defined entirely in terms of themselves. Finally, configuration sets are built to represent each of the states of the parser. The tables are then output to a file for eventual storage in the MSDB.

The LR program and BNF grammar were augmented to allow specification of semantic action routines to be activated during CC. Semantic action tables are produced which list the routines to be executed when a particular language construct is recognized.

The token table generator (scanner generator) algorithm follows the outline provided in *Principles of Compiler Design*, by A. V. Aho, and J. D. Ullman [13]. The basic algorithm parses each regular expression and creates a nondeterministic finite automaton (NFA) to represent it. This NFA becomes part of a larger NFA for all the regular expressions in the file. Once the NFA is complete, it is converted and minimized to a deterministic finite automaton (DFA). The final states of the DFA are matched up with the terminal symbol numbers generated by the LR algorithm. Any action routines to be performed at token recognition are entered into tables, and the token tables are complete and ready for storage in the MSDB during language installation processing.

#### 2.4.2.2.2 Source Instrumentation

The SI formats, preprocesses, and partially instruments the MSIF to identify and flag items such as the following:

- Unit start and end.
- Internal procedures start and end.
- Changes from one language to another (context switches).

Formatting consists primarily of converting tab characters to spaces and indenting the source code to aid in readability of the modified MSIF. Preprocessing is required to eliminate ambiguities in the target software language which would prevent proper functioning of the parser. Preprocessing may also be used to simplify the BNF description of a language by ignoring or simplifying constructs not analyzed by MSAT (e.g., FORTRAN format specifications). Instrumentation is an attempt to reduce the manual effort required to identify certain structures in the TSS.

Automatically generated instrumentation may not be completely accurate. The degree of accuracy in the SI is always dependent upon the TSS source code (i.e., coding standards and consistency of coding techniques), the particular language being scanned, and the degree to which that language can be described in the BNF.

#### 2.4.2.2.3 Data Collection

The DC extracts information on the TSS from an instrumented MSIF. DC processing includes extracting a unit from the MSIF, obtaining the appropriate syntax and semantic action tables from the MSDB, and using the LR(1) parsing technique to identify (perform lexical analysis and parse) language constructs and collect data for storage in the MSDB.

Semantic action routines defined in the BNF for a language are used to affect the manner in which a given construct is treated (e.g., STOP may be counted as a potential singularity, a conditional return may be treated both as a conditional statement for control complexity and as an exit point). The processing of a LOC by the DC is indicated by flags prefixed to each LOC in the annotated source files for each unit. Annotation shows nesting level, language type, statement type, executable/nonexecutable flags, and other items of interest.

#### 2.4.2.3 Static Analysis

The SA functions use the data collected by the SD (TSSI) to calculate and store various software quality metric primitives on a unit-by-unit basis. These may then be combined by the user and/or RG function for the different reports required.

The IOC of MSAT provides static analysis functionality in the following areas:

- a. Complexity measurement. Initially, the calculation of McCabe's Cyclomatic Complexity.
- b. Structure chart preparation. The intermediate processing required prior to structure chart RG.
- c. Error checking. The identification of the following types of errors:
  - Unresolved external references.
  - Units present in the TSS source code but not referenced.
  - Units which call a top-level unit.

d. Interface analysis. Initially, the determination of the number of formal parameters passed by a calling unit which deviate from those expected by the called unit or routine.

e. Standards compliance. The comparison of the metrics of each unit in a TSS (or the TSS cluster metrics) to that system's designated standards criteria.

f. Change analysis. The comparison of one version of a TSS to another version of that same TSS in the following ways:

- Metric compare.
- Structure compare.

g. Statistical profiling (analysis). Data collection and analysis is performed by DC and RG using features of the DBMS (i.e., no SA "statistical analysis" function will be implemented).

#### 2.4.2.4 Report Generation

RG functions produce reports based on the information in the MSDB. RG components usually, but not necessarily, correspond to a related SA component (i.e., the same set of information may be displayed in different form by more than one RG function). Reports produced for the MSAT IOC include the following:

a. Source listing/table of contents (TOC). A source listing containing page/line sequence numbering, which is referenced by the TOC. The TOC is an alphabetized unit list with a unit description (when available) and a reference to the unit's location in the source listing.

b. Software quality metric reports:

(1) Details. One page per unit.

(2) Unit summary. List of all units and their metric values in a columnar format.

(3) Summary. For specified cluster (e.g., TSS, CSC, group of units, etc.).

c. Structure chart. A hierarchical control structure chart showing the nesting level of calls.

d. Error report. A summary of the errors found at a given cluster level.

e. Interface analysis report. A summary of the deviations noted in the number of parameters passed between units for a given cluster level.

f. Standards compliance reports:

(1) Standards exception. A list of units and metrics for those units which were non-compliant (when compared to TSS-specific criteria).

(2) Unit summary compliance. The number and percentage of units which complied for each criterion for that TSS.

(3) System compliance. System- or CSCI-level metrics compared to applicable TSS-specific criteria.

g. Change analysis reports. An analysis of the changes from one version of a TSS to another, to include detailed unit, unit summary, and system change reports for metric and structure data.

#### 2.4.2.5 MSAT Data Base

The purpose of the MSDB is to provide an information storage and retrieval capability for the MSAT system. The MSDB utilizes the INGRES DBMS and the VAX/VMS file management system to accomplish the storage and retrieval of all TSS-related information. INGRES, a relational DBMS from Relational Technology, Inc., is being used to incorporate the advantages of the relational data model into the overall design of MSAT. The use of INGRES provides the flexibility required to facilitate future enhancements, menus and forms generation, report writer capabilities for the creation of user-friendly interfaces and TSS-specific report generation, and a comprehensive query language for aiding the analyst in extracting metrics for software assessments. The VAX/VMS file management system is used to maintain those files which do not lend themselves to storage by the DBMS due to their content and usage.

### 2.5 MSAT Operational Aspects

The following paragraphs provide additional detail on the types of users and their functions which MSAT supports. A synopsis of the operating procedures follows to clarify the operational aspects of the functional components described above.

#### 2.5.1 Personnel Requirements

MSAT supports three types of users. This serves to minimize specialization requirements for the average user while providing a degree of security and data base integrity by limiting access to critical functions. The user categories and typical functions are as follows:

a. Normal user. The normal user is expected to be familiar with logging on and off the VAX/VMS operating system, to have a basic understanding of the VAX/VMS file system and editing functions, and to understand how MSAT output is utilized to evaluate a TSS. This user will typically perform the following types of activities:

- Manually enter TSS-specific information on CSCIs, languages, and standards.
- Instrument TSS source manually and/or via the SI function.
- Invoke the DC function to perform automatic data collection.
- Invoke SA functions.
- Request reports in various formats.

b. Advanced user. The advanced user is expected to have experience in the use of compiler-generator tools, such as parser generators and scanner generators, and to have experience in the specification of a language grammar and semantic actions (augmented BNF grammar). This requires familiarity with the specific source language(s) of a TSS. In addition, the creation of an instrumenter requires familiarity with VAX/VMS FORTRAN. The advanced user will perform all normal user functions, as well as the following:

- Create language-specific instrumenters.
- Create BNF and Regular Expression files for use in the LTG function.
- Add language tables to the MSDB.
- Generate tailored reports via the INGRES Report Writer (requires INGRES and MSDB knowledge.)
- Install new standard documents and their associated criterion.

c. Super user. The MSAT super user is an individual with MSAT "system" privileges. This person must be familiar with the normal and advanced activities, as well as have a complete understanding of the INGRES DBMS and MSAT use of INGRES capabilities. This person is expected to be an MSDB administrator with the capability to grant user privileges and manually manipulate data base tables. The super user might also be an MSAT maintenance programmer, with the privileges required to change the MSDB structure and/or MSAT software. Super users typically perform the following types of activities:

- Archive a TSS to tape and delete it from the MSDB.
- Modify the default SA and RG functions in the menu system.
- Delete language tables currently available in the MSDB.
- Aid users with problems running MSAT.

#### 2.5.2 MSAT Operational Procedures

The purpose of MSAT is to automate the collection of various software design and quality characteristics to support the software assessment of a specific TSS. This includes the automatic extraction of data elements, line/statement counts, and statistics from source code files, the application of various SA functions (e.g., complexity calculations) to the extracted data and counts, and, ultimately, the generation of detailed and summarized reports containing the extracted data and the results of the SA functions. The following paragraphs describe the individual steps involved in the use of MSAT:

a. Convert Source to VAX/VMS Standard Format. The analyst's first step is the conversion of the delivered TSS source to the required input format--an MSIF. An MSIF is a VAX/VMS file which contains ANSI standard ASCII characters, and VAX/VMS end-of-line characters.

b. Initialize MSDB for a New TSS. Prior to processing a new TSS, the analyst must perform a one-time initialization function so that MSAT may create TSS-specific data tables and directories, and update the MSAT information tables. At this point, the analyst may enter TSS-specific software development standards, as well as other TSS and CSCI identification information and language specifications. The MSAT menu/forms interface guides a user through the TSS initialization process.

c. Build New Language-Specific Instrumenter. If MSAT does not possess an automated instrumenter for the desired implementation language, the analyst may either write an instrumenter for that language or choose to manually insert the required MSAT Instrument Lines (MIL). The process of writing an instrumenter is facilitated by some standard shell routines provided by MSAT. After testing the new instrumenter, the analyst uses a menu-driven installation process to make the MSAT system aware of the new instrumenters.

d. Run SI to Automatically Instrument the MSIF. After the analyst has placed the TSS source in the MSIF format and initialized the MSDB for the new TSS, the SI function is executed. This consists of a scanning process which attempts to automatically insert MILs in the MSIF. The capabilities of the instrumenter are language-dependent, but generally this includes marking the start and end of units, the start and end of internal procedures, and language context switches (changes from one language to another within the MSIF).

e. Customize MSIF Instrumentation. After running the instrumenter, the analyst may need to customize the instrumented MSIF by inserting additional instrumentation lines or modifying the SI-inserted MILs to reflect the desired handling of certain source constructs, both intramodule and system level. Other lines may be manually inserted at this time to identify, for example, the begin and end of prologues and/or indicate how particular TSS LOCs should be interpreted (counted) during the execution of the DC function.

f. Manually Instrument the MSIF. Manual insertion of the MILs (via a VAX/VMS editor) is an option available for source languages which have no MSAT instrumenter, or for which writing an instrumenter is less efficient than manual methods. Prior to running DC, the user must have the MSIF in a fully instrumented version, with all required and optional MILs.

g. Create New Language "Parsing" Tables. If MSAT does not possess the semantic, parse, and token tables required to process the TSS implementation language, the analyst must generate new language-description tables for this language. (This effort is simplified and automated by the LTG function; however, the creation of the augmented BNF file and the regular expression file are non-trivial tasks and are not expected of the normal MSAT user.) Once the language table files have been created, the analyst must install this new set of language tables in the MSDB. An MSAT menu aids the user in the installation of new language table files in the MSDB, thus providing the user (and all subsequent users) with the capability to process source code in that language.

h. Run DC to Perform Data Collection. The DC function scans the instrumented MSIF, and collects software metric data on a unit-by-unit basis. An annotated source file is created at this time for each unit and saved within the MSAT/TSS directories for later report generation. (Any instrumentation anomalies are brought to the attention of the user during the collection process. An analyst may then perform another iteration of the customize-MSIF-and-run-DC to collect the data as desired for the subsequent SA and RG functions.)

i. Run SA Functions. After the collection pass has been successfully completed, the analyst requests the particular SA functions required for the associated reports. The most commonly desired SA functions are available as a default set to simplify this task.

j. Generate Reports. After the SA functions have executed, any of a variety of reports may be generated. If a non-standard report format or query against the TSS data in the MSDB is desired, the analyst may use the manual data base entry menus and/or the INGRES Report Writer to create a report tailored to the specific situation.

## 2.6 Future Development

The IOC of MSAT was designed to provide the essential functions required of a static analysis tool in the given environment. An integral part of the design included provisions for P<sup>3</sup>I, graphically portrayed in figure 7. Some P<sup>3</sup>I tasks have already been identified while others will require further methodological investigation.

### 2.6.1 Candidate Tasks for MSAT P<sup>3</sup>I

The tasks described below represent additions to the IOC of MSAT and are achievable with current technology as demonstrated by existing tools. Although future functionality will be dependent upon prioritization by TECOM I/FOAs, other users, and feedback from application of MSAT, some recommended additions to the current capabilities are possible based on present knowledge. A partial list of candidate tasks for future enhancement of MSAT follows:

a. Creation of a language library for a robust language description of each major language category likely to be encountered. Subsequent additions to the library would consequently be subsets or minor variations (dialects) of existing descriptions, resulting in considerable savings for implementation of new languages.

b. Augmentation of automatic SI. Automatic prologue, entry point, etc. instrumentation would increase efficiency over manual methods.

c. Prologue processor capability (collecting and reporting statistics on items within target software prologues). Manual assessment is rarely performed because of cost. Approximately 30 to 1 increase in efficiency may be expected by automating.

d. Optimization of SI/DC processing. The goal would be to reduce wall clock processing time to effect a similar savings in analyst "idle" time. One area offering considerable savings is reducing the number of passes and copies of the MSIF.

e. Software metric enhancement. Addition of Halstead's software science measures (e.g., program length) and software modification assessment metrics would provide more accurate and efficient software quality parameters. These parameters would enhance the objectivity and consistency of software assessment.

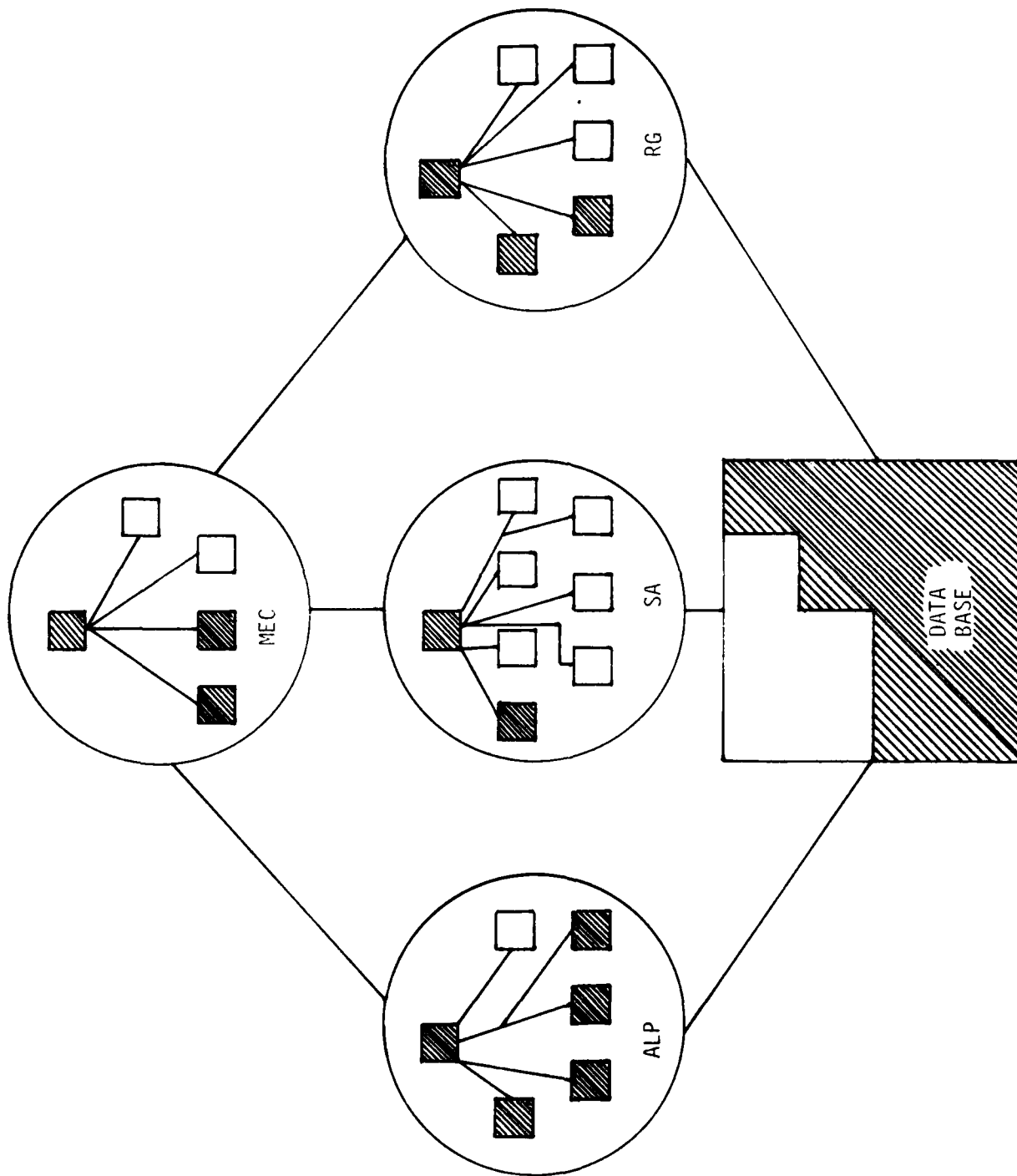


Figure 7. Pre-Planned Product Improvement

f. Assembly language construct processing. Because assembly language typically comprises 40 percent of software tested at USAEPG, greater efficiency in testing would result from enhancing assembly language processing capability (e.g., macro/conditional assembly language, equates/include files, entry point/ internal routine definitions, and indirect addressing/interrupt service routine handling.)

g. Source compatibility preprocessor. Creation of a library or tool bench of reusable software would aid in converting foreign (non-VAX) file formats to an MSAT/VAX-compatible form. Examples are: ROLM to VAX tape conversion, intelligent/table-driven editor, symbol definition/substitution (for include/copy or conditional code processing), etc.

h. Software performance/reliability capability. This would result in expanding the static software parameters in the MSAT data base to include dynamic performance parameters. Initially, this would include reliability information (software Test Incident Reports) to satisfy the requirements for performing the standard software maturity subtests.

i. Sizing/timing information. The MSAT data base could be expanded to include sizing and timing information. Sizing information, in conjunction with software reliability information, is required to address reliability per DOD-STD-1679A. Timing information can provide valuable information for examining critical functions in time-sensitive applications.

j. Text compression. The most significant amount of textual information used by MSAT resides in the TSS source code (MSIF) and annotated source files. Text compression techniques used by some document processing systems provide a method to reduce the storage requirements of the annotated source contained in the MSDB.

#### 2.6.2 Software Test Methodology

It is well known that the software testing arena suffers from a lack of quantitative, generally accepted test methods and criteria. The current methodology is sufficiently mature to provide some results for evaluation, but is largely inadequate from a theoretical viewpoint. A major deficiency is the absence of systematic methods for identifying critical functions and level of test (thoroughness), not presently addressed quantitatively by test and evaluation guidance [14].

Since the software testing discipline lacks maturity, a characteristic shared by software development in general, continual monitoring of advances in the technology is required. As new techniques are developed to a stage of practical applicability, they should be incorporated into tools such as MSAT. Significant technological progress should be accompanied by reevaluation and revision of the software test methodology to maintain currency of the test and evaluation process.

APPENDIX A  
METHODOLOGY INVESTIGATION PROPOSAL

PREVIOUS PAGE  
IS BLANK

June 1984

## METHODOLOGY INVESTIGATION PROPOSAL

1. TITLE. Multilingual Static Analysis Tool.
2. CATEGORY. VISTA, DC3I, SMI/Software, Interoperability.
3. INSTALLATION. U.S. Army Electronic Proving Ground, Fort Huachuca, Arizona 85613.
4. PRINCIPLES INVESTIGATOR. Mr. Richard G. Jacques, Software and Automation Branch, STEEP-MT-DA, AUTOVON 879-1957.
5. STATEMENT OF THE PROBLEM. The techniques and measures of performance (MOPs) to aid an analyst in assessing the quality features of a software system under test (SUT) have evolved to the point of practical application. TECOM has such an application tool, the Program Flow Analyzer (PFA). The application, however, has to evolve in the same manner that software and C<sup>3</sup>I technology are evolving to produce a design that requires less resources to tailor to different host processors/languages expected to be tested by TECOM, and be user-friendly in identifying and providing the desired reports to the analyst.
6. BACKGROUND. The PFA was conceived under TECOM Project No. 7-CO-RD7-EP1-001, Position Location Reporting System (PLRS) Software Test Methodology. An A-level specification was developed which provided the requirements for a PFA. The PFA design, coding, testing, and documentation was performed under TECOM Project No. 7-CO-RD0-EP1-004. The DEC-10 system and SNOBOL type derivative language was used as the implementing host processor and language. The PFA concept and utility was validated on the PLRS and Tactical Computer System (TCS) DT II software evaluations. The time required to generate the processor/language-specific (front end) portion is estimated to take from two to six man-months of effort, based on complexity of the software SUT. The PFA methodology plus elements of a new concept to shorten the front end development time were applied to the USAEPG's Integrated Inertial Navigation System (IINS) program and to the SGT. York Fire Control Computer (FCC) program in support of USAAPG's overall software assessment effort. The new concept shortens the development time of the front end. In addition, the PFA has evolved to more than a flow analysis tool; it is a static software analysis tool. The software and C<sup>3</sup>I technology evolution, life cycle software support center's (LCSSCs), and pre-planned product improvement (P<sup>3</sup>I) processes require a software test tool that is tailored to the Developer-Tester community. Hence the new name and acronym, Multilingual Static Analysis Tool (MSAT).
7. GOAL. Develop a software analysis tool with initial implementation on the VAX 11/780 using the concepts proven by the PFA. Sub-goals of this investigation are:
  - a. Develop MSAT which will greatly reduce the time and effort to create the processor and language-specific front end.
  - b. Develop a user-friendly, interactive, man-machine interface to the MSAT report writers which will assist the software analyst in obtaining the desired output reports.

c. Incorporate new metrics as identified by the PFA methodology into MSAT.

d. Validate and configuration manage the MSAT to the development and test community.

## 8. DESCRIPTION OF INVESTIGATION

a. The U.S. Army Electronic Proving Ground will take the concepts, results, and experience from TECOM Project No. 7-CO-RDO-EP1-004, Program Flow Analyzer, and design, implement, test, and document a computer program to assist a software analyst in assessing the quality features of the software system under test.

b. USAEPG will:

(1) Coordinate with other I/FOAs as to their unique requirements for a software analysis tool (MSAT).

(2) Coordinate with other I/FOAs on the design of MSAT by having the I/FOAs actively participate in quarterly program reviews which will be held in conjunction with the TECOM Software Technical Committee (TSOTEC) meetings.

(3) Provide technical and program management directions for MSAT development to best meet the needs of USAEPG and other I/FOAs in satisfying test project workloads for software quality assessments.

(4) Design, code, test, and document MSAT.

(5) Train appropriate personnel (EPG, other I/FOA, and other personnel) in the use of MSAT.

(6) Sustain, maintain, and perform configuration management of MSAT.

c. Investigation Schedule.

Milestone/Phase	Schedule							
	FY 84 (Qtrs)				FY 85 (Qtrs)			
	1	2	3	4	1	2	3	4
MSAT Requirements Definition (SRR)				X				
Preliminary Design Review					X			
Critical Design Review						X		
MSAT Coding				X	X	X	X	
Initial Support to Projects						X		
MSAT Documentation			X	X	X	X	X	X
Software Requirements Specification			X	X				
B-5 or Equivalent			X	X	X			
C-5 or Equivalent				X	X	X		
Maintenance							X	X
User's Manual							X	X

Milestone/Phase	Schedule							
	FY 84 (Qtrs)				FY 85 (Qtrs)			
	1	2	3	4	1	2	3	4
Configuration Management/P <sup>3</sup> I								X
Training								X
Project Final Report								X

d. This investigation will result in a more capable, more efficient, user-friendly, and transportable software analysis tool.

e. Environmental Impact Statement. The execution of this task will not have an adverse impact on the quality of the environment.

f. Health Hazard Statement. No health hazards are anticipated.

## 9. JUSTIFICATION

### a. Mission and Impact Statements.

(1) Association with mission. The USAEPG/TECOM mission includes the responsibility to conduct software testing of systems containing embedded computer resources. Software testing includes the assessment of the quality of the software for post-deployment supportability. MSAT is a tool usable by the analyst to meet this test requirement. The proportion of systems being developed which contain embedded computer resources is growing. Current estimates are that USAEPG will have test responsibility for over 120 of the 166 systems containing embedded resources that are under development within DARCOM, with others being tested by other TECOM I/FOAs.

(2) Present Capability, Limitations, Improvement, and Impact on Test if not approved. The PFA is a general purpose software analysis tool. USAEPG project officers have identified the need to develop as many as 12 different language/processor front ends. USAEPG has used the results from application of the PFA methodology (USAEPG support to USAAPG for the SGT YORK FCC software assessment) and realizes the value of such a tool. USAAPG has informed USAEPG of the desire to use PFA on the other SGT YORK processors and several training simulator devices (exact number of front ends is unknown, however, each system usually has two front ends, one for the HOL, and one for the assembly routines). The personnel and the time required to understand PFA to generate the necessary front ends are not available; a faster means to generate the front ends and more user-friendly report writers must be created in order to adequately support the test workload, hence the need to develop MSAT.

b. Dollar Savings. Manpower savings using MSAT is estimated on the average ratio of 30:1. Some of the reports provided by MSAT would not normally be attempted manually because of the sheer volume and inherent mistakes which are often made.

c. Workload. The following Army Battlefield Automated Systems are examples of systems under development which are programmed for testing by TECOM during the timeframe shown.

System	Test Schedule (FY)					
	85	86	87	88	89	90
JTIDS	X	X				
MCS	X			X	X	X
RPV	X	X		X	X	
PLRS		X	X			
DTSS			X	X	X	
SHORAD C2			X	X	X	
JINTACCS			X	X	X	X
Improved GUARDRAIL V	X					
REGENCY NET	X					
PJH	X	X	X	X		
GPS	X	X	X	X		
ASAS	X	X	X	X		
FIREFINDER		X	X	X	X	
HAWK PHASE III	X					
SHORAD C2		X	X			
AN/TSQ-73 Software Benchwork	X	X	X	X	X	
PATRIOT Growth Program	X	X	X	X	X	

d. Recommended TRMS Priority. Refer to the workload paragraph (10c) and the ODCSOPS priority listing. This project supports the DOD STARS initiative, and DADAS Letter, Development Testing of C<sup>3</sup>I, dated 1 August 1983.

e. Association with Requirements Documents. The requirement for this methodology is not derived from the requirements documents associated with specific material developments. The requirement is identified, however, by:

- (1) Findings of the Army Science Board.
- (2) DOD STARS program.
- (3) ABIC, AC2MP, JINTACCS.

to develop automated software tools to facilitate the fielding of software in embedded computer systems.

f. Others. None.

## 10. RESOURCES.

a. Financial.

- (1) Funding Breakdown.

	Dollars (Thousands)			
	FY 84		FY 85	
	In-House	Out-of-House	In-House	Out-of-House
Personnel Compensation	8		6	
Travel	1.5		1.5	
Contractual Support		100.0		219.0
Academia				
Materials & Supplies	0.5		0.5	
ADP			2.0	
Subtotals	10.0	100.0	10.0	219.0
FY Totals	110.0		229.0	

(2) Explanation of Cost Categories.

(a) Personnel Compensation. Pay of in-house personnel assigned to the investigation.

(b) Travel. Task coordination and technical liaison with support contractor.

(c) Contractual Support. A major portion of the investigation will be accomplished by tasking the USAEPG Software/Interoperability Support Contractor.

(d) Materials and Supplies. Routine support materials.

(e) ADP. ADP utilization for the generation of software.

b. Anticipated Delays. None.

c. Obligation Plan (FY 84).

FY QTR	1	2	3	4	Total
Obligation Rate (\$K)			105.0	5.0	110.0

d. In-House Personnel.

(1)

	FY 85		
	Manhours		
	Number	Required	Available
Elect Engr, GS-0855	1	300	300
Comp Sci, GS-0334	1	100	100

(2) Resolution of Non-Available Personnel. Required in-house personnel are expected to be available.

11. INVESTIGATION SCHEDULE (FY 84).

	O	N	D	J	F	M	A	M	J	J	A	S
In-House	-	-	-	.	.	.	.	.	1	.	.	2
Contract	-	-	-	-	-	-	-	-	-	-	-	-

Symbols: - - - Active investigation work (all categories).

. . . Contract monitoring (in-house only).

1 Interim report to HQ TECOM.

2 Interim report to HQ TECOM.

12. ASSOCIATION WITH TOP PROGRAM. TOP 1-i-056, Software Testing, will need to be revised as a result of this investigation.

FOR THE COMMANDER:

(signed)

. MELVIN FOWLER  
LTC, SigC  
Director of Materiel Test

APPENDIX B  
REFERENCES

## REFERENCES

1. Methodology Investigation Final Report Program Flow Analyzer, dated January 1984. TECOM Project No. 7-CO-RD0-EP1-004. U.S. Army Electronic Proving Ground, Fort Huachuca, Arizona 85613.
2. Methodology Investigation Final Report PLRS Software Test Methodology, dated 4 April 1980. TECOM Project No. 7-CO-RD7-EP1-004. U.S. Army Electronic Proving Ground, Fort Huachuca, Arizona 85613.
3. Methodology Investigation Final Report Program Flow Analyzer, dated 30 October 1982. TECOM Project No. 7-CO-RD0-EP1-004. U.S. Army Electronic Proving Ground, Fort Huachuca, Arizona 85613.
4. Letter Report, Program Flow Analyzer (PFA) for Air Defense Weapon Computer Processors, TECOM Project No. 7-CO-RD3-EP1-005, STEEP-MT-DA, 19 October 1983.
5. Federal Software Exchange Catalog, PB85-904001, National Technical Information Service (NTIS), General Services Administration (GSA), 1985.
6. "Software Development Tools", NBS Special Publication 500-88, National Bureau of Standards (NBS), 1982.
7. Software Engineering Automated Tools Index, Software Research Associates, San Francisco, California, December 1982.
8. OSD/DDT&E Software Test and Evaluation Project (STEP) Final Report, Volume 2, Software Test and Evaluation: State-of-the-Art, OSD/DDT&E, Washington, D.C., Georgia Institute of Technology, Atlanta, Georgia, June 1983.
9. Automation of Quality Measurement, Final Technical Report, RADC-TR-82-247, Rome Air Development Center, Griffiss AFB, NY, U.S. Army Computer Systems Command, Georgia Institute of Technology, Atlanta, Georgia, General Electric Company, September 1982.
10. A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE), NBSIR 82-2625, National Bureau of Standards, Washington, D.C., February 1983.
11. "LR-Automatic Parser Generator and LR(1) Parser," Wetherell, Charles and Shannon, Alfred, IEEE Transactions on Software Engineering, Vol. SE-7, No. 3, May 1981.
12. "A Practical General Method for Constructing LR(k) Parsers," Pager, David, Acta Informatica, Vol. 7, 1977.
13. Principles of Compiler Design, Aho, A.V. and Ullman, J.D., Addison-Wesley, 1977.
14. Policy Recommendations for Software Test and Evaluation: System Level Test Issues, DeMillo, R.A., et al, Software Test and Evaluation Project (STEP) Georgia Institute of Technology.

APPENDIX C  
ACRONYMS AND ABBREVIATIONS

PREVIOUS PAGE  
IS BLANK

## ACRONYMS AND ABBREVIATIONS

ABIC..... Army Battlefield Interface Concept  
AC<sup>2</sup>MP..... Army Command and Control Master Plan  
ADP..... Automatic Data Processing  
AI..... Artificial Intelligence  
ALP..... Automated Language Processing  
ALS..... Ada Language System  
AMC..... U.S. Army Material Command  
AMT..... Automated Measurement Tool  
APSE..... Ada Programming Support Environment  
ASAS..... All Source Analysis System  
ASCII..... American Standard Code for Information Interchange  
ASM..... Assembly  
ATLAS..... Abbreviated Test language for All Systems  
BNF..... Backus-Naur Form  
C<sup>3</sup>I..... Command, Control, Communications, and Intelligence  
CDR..... Critical Design Review  
COBOL..... Common Business Oriented Language  
Comp..... Computer  
CONOPS..... Concept of Operation  
CPC..... Computer Program Component  
CSC..... Computer Software Component  
CSCI..... Computer Software Configuration Item  
DACS..... Data and Analysis Center for Software  
DARCOM..... U.S. Army Materiel Development and Readiness Command (now AMC)  
DBDD..... Data Base Design Document  
DBMS..... Data Base Management System  
DC..... Data Collection  
DC<sup>3</sup>I..... Distributed C<sup>3</sup>I  
DDT&E..... Director Defense Test and Evaluation  
DEC..... Digital Equipment Corporation  
dept..... Department  
DFA..... Deterministic Finite Automaton  
DIVAD..... Division Air Defense  
DoD..... Department of Defense  
DT..... Developmental Test  
DTIC..... Defense Technical Information Center

PREVIOUS PAGE  
IS BLANK

DTSS..... Digital Topographic Support System  
 Elect..... Electronic  
 Engr..... Engineer  
 EPG..... (see USAEPG)  
 EQUQL..... INGRES Embedded Query Language  
 FCC..... Fire Control Computer (Sgt. York DIVAD)  
 Fortran..... Formula Translation  
 FQT..... Formal Qualification Test  
 FY..... Fiscal Year  
 GPS..... (see NAVSTAR GPS)  
 GSA..... General Services Administration  
 HOL..... High-Order Language  
 HQ..... Headquarters  
 IEEE..... Institute of Electrical and Electronics Engineers, Inc.  
 I/FOA..... Installation/Field Operating Activity  
 IINS..... Integrated Inertial Navigation System  
 INGRES..... Interactive Graphics and Retrieval System  
 I/O..... Input/Output  
 IOC..... Initial Operational Capability  
 JINTACCS..... Joint Interoperability of Tactical Command and Control Systems  
 JTIDS..... Joint Tactical Information Distribution System  
 LCSEC..... Life Cycle Software Engineering Center  
 LCSSC..... Life Cycle Software Support Center (now LCSEC)  
 LISP..... List Programming Language  
 LOC..... Line of Code  
 LR..... Left-to-Right (Parser)  
 LR(1)..... Left-to-Right With (One) Lookahead  
 LTC..... Lieutenant Colonel  
 LTG..... Language Table Generation  
 MCS..... Maneuver Control System  
 MEC..... MSAT Executive Control  
 MIL..... MSAT Instrument Line  
 MOP..... Measures of Performance  
 MSAT..... Multilingual Static Analysis Tool  
 MSDB..... MSAT Data Base  
 MSIF..... MSAT Standard Input File  
 NAVSTAR GPS... Global Positioning System

NBS..... National Bureau of Standards  
 NFA..... Nondeterministic Finite Automaton  
 NTIS..... National Technical Information Service  
 ODCSOPS..... Office of the Deputy Chief of Staff, Operations  
 OSD..... Office of the Secretary of Defense  
 P<sup>3</sup>I..... Pre-Planned Product Improvement  
 PDR..... Preliminary Design Review  
 PFA..... Program Flow Analyzer  
 PJH..... PLRS/JTIDS Hybrid  
 PLRS..... Position Location Reporting System  
 PROLOG..... Programming in Logic  
 QTR..... Quarter  
 RADC..... Rome Air Development Center  
 RG..... Report Generation  
 RPV..... Remotely Piloted Vehicle  
 Rqmts..... Requirements  
 SA..... Static Analysis  
 Sci..... Scientist  
 SD..... Source Decomposition  
 SDF..... Software Development Folder  
 SDP..... Software Development Plan  
 SDS..... Software Development Standard  
 SHORAD C<sup>2</sup>..... Short-Range Air Defense, Command and Control  
 SI..... Source Instrumentation  
 SMI..... Soldier Machine Interface  
 SNOBOL..... String-Oriented Symbolic Language  
 SPM..... Software Programmer's Manual  
 SRR..... Software Requirements Review  
 SRS..... Software Requirements Specification  
 SSS..... System Segment Specification  
 STARS..... Software Technology for Adaptable Reliable Systems  
 STEP..... Software Test and Evaluation Program  
 STLDD..... Software Top-Level Design Document  
 STP..... Software Test Plan  
 SUM..... Software User's Manual  
 SUT..... System Under Test  
 TCS..... Tactical Computer System

TECOM..... U.S. Army Test and Evaluation Command  
TLCSC..... Top-Level CSC  
TOC..... Table of Contents  
TOP..... Test Operations Procedure  
TRMS..... Test Resource Management System  
TSOTEC..... TECOM Software Technical Committee  
TSS..... Target Software System  
TSSI..... TSS Information  
USAAPG..... U.S. Army Aberdeen Proving Ground  
USAEPG..... U.S. Army Electronic Proving Ground  
VAX..... Virtual Address Extension  
VHOL..... Very High Order Language  
VISTA..... Very Intelligent Surveillance and Target Acquisition System  
VMS..... Variable Message System

APPENDIX D  
SOFTWARE HIERARCHY DEFINITIONS

1.0 Scope. The following paragraphs, and figure 8, identify and define the terms which shall be used to describe the MSAT software hierarchy, as well as the software hierarchies of the SUTs which are processed by MSAT.

#### Software System

A combination of associated CSCIs and computer data required to enable the computer hardware to perform computational or control functions.

#### Computer Software Configuration Item

An aggregate of computer software which satisfies an end use function and is designated for configuration management.

#### Computer Software Component

A functional or logically distinct subset of a CSCI, consisting of one or more units or CSCs.

#### Unit (Module)

The lowest level logical entity specified in the detailed design which completely describes a non-divisible function in sufficient detail to allow implementing code to be produced and tested independently of other units. Units may consist of one or more routines.

#### Routine

A set of instructions and/or statements that exist as an identifiable entity and carry out some well defined operation or set of operations. A routine is usually the smallest compilable element of a software system. The terminology for this, and lower levels, of the software hierarchy varies with the particular language. Terms which may be synonymous are procedure, subroutine, function, subprogram, package, etc.

#### Block

A sequence of statements that are well defined for block-structured languages but are less appropriate for other languages. Other entities which may appear at this level of the hierarchy, but are language-dependent, are: internal procedure, loop and case constructs, etc.

#### Segment

A logical segment (or decision-to-decision path) is the set of statements in a program which are executed as the result of the evaluation of some predicate (conditional) within the program. The segment should be thought of as including the sensing of the outcome of a conditional operation and the subsequent statement execution up to and including the computation of the next predicate value, but not including its evaluation.

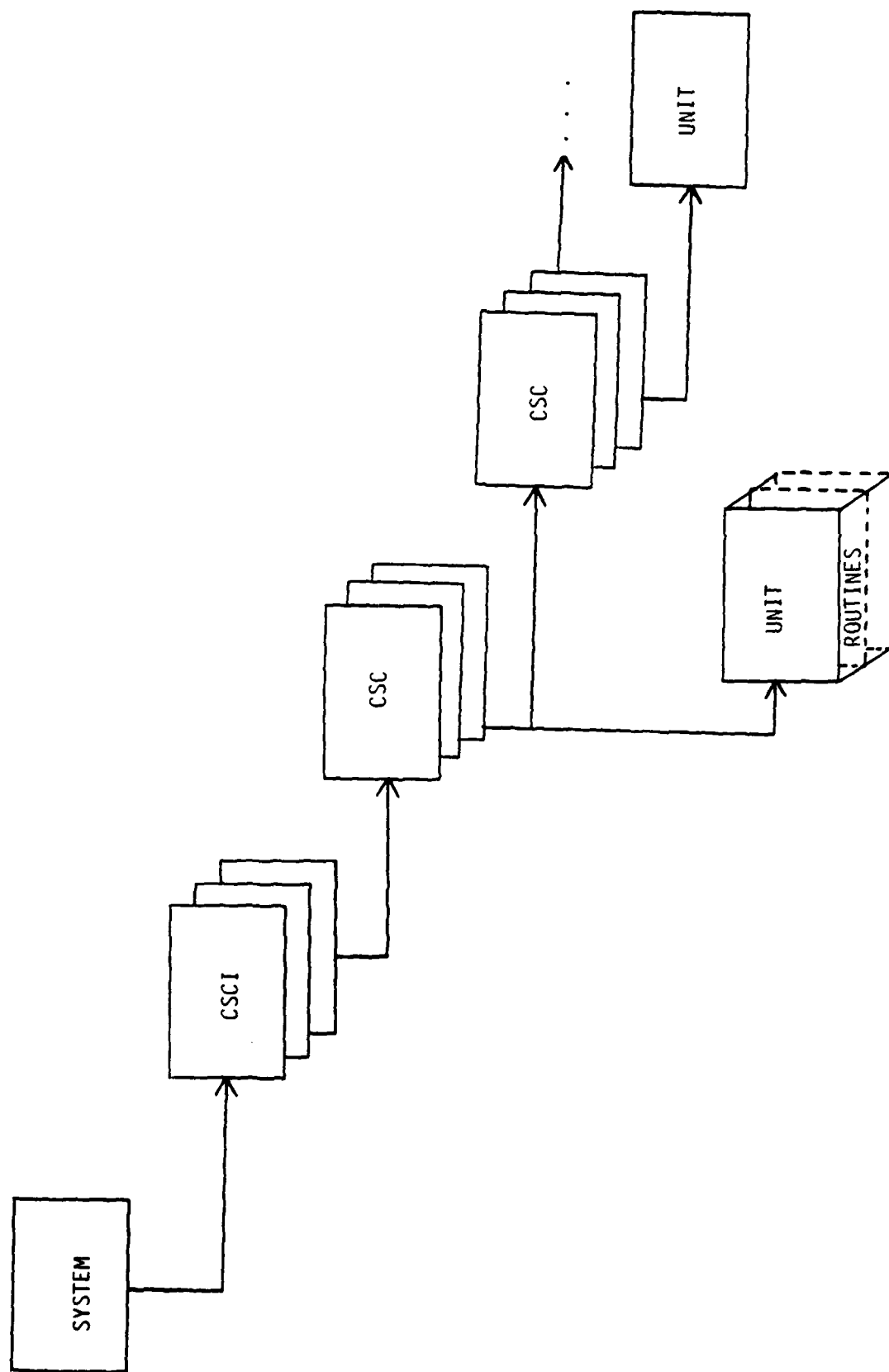


Figure 8. Software Hierarchy

### Statement

An instruction or set of lexical elements delineated by the syntax of a particular language. Executable statements cause some action to be performed.

### Line of Code

A string of characters contained in one logical record of the program source. Usually, one LOC contains one statement. The concept of "LOC" is used to identify lines which contain multiple statements and to supplement statement level metrics where a statement may be composed of multiple lines of code.

APPENDIX E  
MSAT GLOSSARY

PREVIOUS PAGE  
IS BLANK

1.0 Scope. The following terms are identified and defined as they are used throughout the MSAT software development process. Some of the terms are included to standardize their usage with respect to the software attributes collected, metrics generated, and static analysis functions performed by MSAT.

#### Annotated Source File

An annotated source file is created for each unit found in the MSIF. The annotated source file contains line numbers and various flags which are indicative of the way each particular source line was interpreted by the parser.

#### Auditing

Conducting examination to determine whether or not predefined rules have been followed.

#### Augmented BNF Grammar File

Contains the source language descriptions required by the LTG for the generation of a parser that will recognize the language. This BNF grammar is augmented by statements that indicate what semantic action(s) are to be taken in the event a given construct within the language is recognized. The BNF grammar is the input to the LR(1) parse table generator.

#### Block

A sequence of statements acting or regarded as a single entity.

#### Cluster

Any TSS-defined (e.g., CSCI) or user-defined (e.g., a list of certain units) logical grouping of units for the purposes of report generation or manual data base retrieval.

#### Cohesion

A measure of the strength of association of the elements within a module.

#### Comment

A lexical element used to annotate a program. Comments usually have no affect upon proper execution of the software. "Comment" usually refers to a non-blank LOC which contains a comment statement or a portion thereof.

#### Comparison

Determining and assessing similarities between two or more items. In particular, performing change analysis on two versions of the same computer program to identify changes in the source code, documentation, or hierarchical structure.

PREVIOUS PAGE  
IS BLANK

### Completeness Checking

Assessing whether or not an entity has all its parts present and if those parts are fully developed. A tool that examines the source code for missing parameter values has this feature.

### Complexity Measurement

A method of determining how complicated an entity is (e.g., model ... system) by evaluating some number of associated characteristics.

### Computer Software Component

A functional or logically distinct subset of a CSCI, consisting of one or more units or CSCs.

### Computer Software Configuration Item

An aggregate of computer software which satisfies an end use function and is designated for configuration management.

### Consistency Checking

The determination of whether or not an entity is internally consistent in the sense that it contains uniform notation and terminology, or is consistent with its specification. For example, checking for consistent usage of variable names or consistency between design specifications and code.

### Construct

A statement or set of related statements, e.g., the five structured programming control structures: SEQUENCE, IF-THEN-ELSE, CASE, DO-WHILE, DO-UNTIL.

### Coupling

A measure of the interdependence of modules in a design structure; the type of data and control shared between two modules.

### Cross-Reference

Referencing entities to other entities by logical means. In particular, a cross-reference could illustrate all the variables and routines referenced by a unit.

### Data Flow Analysis

A graphical analysis of the sequential patterns of definitions and references of data.

### Decision-to-Decision Path

See segment.

### Decision Node

A node in a directed flow graph which corresponds to a decision statement within the source code.

### Default Standards

Predefined standards (e.g., MIL-STD-1679A) which may be used for determining compliance with the standard in lieu of or in addition to user-defined standards.

### Directed Graph

A directed graph (digraph) consists of a set of nodes which are interconnected with oriented arcs. A program digraph normally has only one entry and one exit node.

### Entity

Anything that can be named in a program. A hierarchy of entities and constituent elements may be defined.

### Error Checking

The determination of discrepancies, their importance, and/or their cause, e.g., identification of possible program errors, such as misspelled variable names, arrays out of bounds, and modifications of a loop index.

### Executable Statement

A statement which causes some action to be performed, as opposed to a declaration which defines an entity.

### Expandability

Those attributes of software that provide for increased data storage or computational functional capability.

### External Reference

A list of references supplied by the user which are external to the target software source, which would otherwise be identified by MSAT as undefined references (e.g., operating system library routines).

### Fan-In

The fan-in of a module is the number of distinct modules that call this module (e.g., the number of modules that are immediately superordinate to this module).

### Fan-Out

The fan-out of a module is the number of distinct modules that are called by this module (e.g., the number of immediately subordinate modules).

### Hierarchical Structure Design

A design method in which interactions between modules are restricted to flow of control between a predecessor module and its immediate successor modules.

### Hierarchy Level (or Cluster)

The grouping of units or a hierarchy level (System, CSCI, etc.) desired for a specific request.

### Identifier

One of the basic lexical elements of a language. An identifier may be used as the name of an entity or as a reserved word.

### Instrumented MSAT Standard Input Files

The modified MSIF contains updated (or new) instrumentation from the SI process or manual insertion.

### Interface Analysis

The checking of the interfaces between program elements for consistency and adherence to predefined rules and/or axioms. In particular, checking parameter usage (type, number) in calling and called routines. Determining the various degrees of module coupling might also be included in interface analysis.

### Input/Output Specification Analysis

The analysis of the input and output specifications in a program usually for the generation of test data.

### Language Dialect

The particular dialect of a language version (e.g., Singer-Kearfott FORTRAN).

### Level (Detail, Summary)

The level, as used here, refers to the level of report which is desired in a particular user request (e.g., detailed or summarized unit level software quality metrics).

### Library

A collection of routines (or data) which are frequently used (e.g., I/O, SINE) and are externally referenced or included in the software being developed.

### Line of Code

A string of characters contained in one logical record of the program source. Usually, one LOC contains one statement.

### Log File

A history of the actions performed (summary statistics) and error messages generated during the execution of a given function or set of functions.

### LR

LR is a pair of programs--an automatic parser generator and an LR(1) parser. LR uses a powerful algorithm to generate a space efficient parser for any LR(1) (left to right with (1) lookahead) grammar. The parser generator reads a context-free grammar in a modified BNF format and produces tables which describe an LR(1) parsing automaton. The parser is a set of subroutines that interpret the tables to parse an input stream of tokens supplied by a (locally written) lexical analyzer.

### Manual Data Entry

Data which must be entered manually into the data base; for example, data which cannot be collected automatically, or data collected which must be supplemented or modified prior to analysis.

### Menu Options

Menu selections which occur during the MEC interaction with a user. The user selects an available option and the MSAT MEC acts upon this selection.

### Meta-Language

A meta-language is a set of symbols and words used to describe another language (in which these symbols do not appear). The most common application is in the definition of programming languages (e.g., BNF grammars).

### Metrics

Software metrics produced by the SA function; includes summaries statistics, counts, etc.

### Module

An independently compilable software component. The term "module" is frequently used in industry to be synonymous with "unit" as defined herein.

### MSAT Data Base

Includes VAX/VMS files and INGRES data tables. The MSDB contains TSSI, including software attributes and annotated source.

### MSAT Standard Input File

MSAT expects a standard format for its input files, the "MSIF" in this document implies any user specified input file containing TSS source in the MSAT format. This file may contain one or more units of TSS source code.

### Parse Tables

These tables are used with a parser and lexical analyzer to recognize the source code input to MSAT. The tables give the parser the "knowledge" to recognize a production within the grammar or determine that another symbol is required before such a recognition can occur.

### Path Segment

See segment.

### Process

The transformation of input data flow(s) into output data flow(s).

### Routine

A set of instructions and/or statements that exist as an identifiable entity and carry out some well defined operation or set of operations. A routine is usually the smallest compilable element of a software system. The terminology for this, and lower levels, of the software hierarchy varies with the particular language. Terms which may be synonymous are: procedure, subroutine, function, subprogram, package, etc.

### Scanning

Examination of an entity sequentially to identify key areas or structure. For example, examining source code and extracting key information for generating documentation or source analysis.

### Segment

A logical path segment (or Decision-to-Decision Path) is the set of statements in a module which are executed as the result of the evaluation of some predicate (conditional) within the program.

### Software Quality

The composite of attributes, including performance, which describe the degree of excellence of the software; features and characteristics of a software product or a related service to satisfy a given need.

### Software System Independence

Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, I/O routines, etc.).

### Source (Language) Description

The language grammars and prologue format descriptions, (e.g., modified BNF for HOL and ASM).

### Statement

An instruction or set of lexical elements delineated by the syntax of a particular language.

### Static Analysis

Examining the source code statically (not under execution conditions) and performing syntax analysis, structure checks, module interface checks, event sequence and analysis, and other similar functions.

### Statistical Analysis

Performing statistical data collection and analysis on software source code.

### Structure Checking

Detecting structural flaws within a program (e.g., improper loop nestings, unreferenced labels, unreachable statements, and statements with no successors).

### Subroutine

An independently compilable sequence of statements that performs a specific function (usually used to mean "external" routines or routines belonging to a library; see also routine).

### System

A combination of associated computer programs and computer data required to enable the computer hardware to perform computational or control functions.

### Token Tables

The token tables contain information used to detect all the terminal symbols of the language, that is, those which cannot be further reduced. Such symbols are keywords, numerals, operators, and identifiers. These tables allow the lexical analyzer to recognize the terminal symbols of the language and pass this information to the parser.

### Trace File

A trace file is produced whenever a user requests either the brief, verbose, or test trace mode. This file contains detailed information tracing the activities of the major MSAT CSCs. It is used primarily for detecting BNF/language table problems, inaccurate or missing instrumentation in an MSIF, and aiding in MSAT development and/or enhancement.

### TSS Hierarchy Terminology

The software hierarchy terminology used by a specific TSS for report generation. (The default is DOD-STD-SDS terminology: system, CSCI, CSC, and unit.)

### TSS Source

The original (raw) software source code file which will be transformed into the MSAT standard input format. The source code may be composed of HOL with embedded ASM, embedded VHOL, both, or may consist of a single language.

### Type Analysis

The evaluation of whether or not the domain of values attributed to an entity are properly and consistently defined.

### Unit

The lowest level logical entity specified in the detailed design which completely describes a non-divisible function in sufficient detail to allow implementing code to be produced and tested independently of other units. (A unit may be made up of several routines.)

### Units Analysis

The determination of whether or not the units or physical dimensions attributed to an entity are properly defined and consistently used ensuring variables used in computations have proper units (e.g., hertz = cycles/second).

### Unreachability

A statement (or segment) is unreachable if there is no logically obtainable set of input data which can cause the statement (or segment) to be traversed.

### User Input Parameters

The data and control information which will be input by the user through the MSAT user interface.

### User Standards

Standards (criteria values) input by the user to be used for standard's compliance, e.g., TSS-specific standards. A user might also specify a specific set of predefined standards which will be retained for reference in the MAST data base: MIL-STD-SDS, MIL-STD-1679A, etc.

APPENDIX F  
SOFTWARE STANDARDS COMPARISON

STANDARD:		MIL-STD-1679 Dec 1978	DOD-STD-1679A 22 Oct 83	DOD-STD-SDS (Proposed) 5 Dec 83
1. Module Size	Each $\leq$ 200 average 100 (5.3.7)	Each $\leq$ 200 average $\leq$ 100 (same) (5.3.6)	Same, specifies "non- expandable" each $\leq$ 200 average $\leq$ 100 (20.2.3)	
2. Control Structures	5 basic: Sequence, If-then-Else, Do-While, Do-Until, Case (5.3.1)	Same 5 basic: (5.3.2)	Same, permissible to simulate or use precompiler (20.2.1.2) (20.2.2) 5 structures	
3. Included/Copied Segments	In HOL only (5.3.2) used for repetitive segments (5.4.5.2)	Same (5.3.3) Repetitive segment use (5.4.15)	Not included	
4. Entry-Exit	Single entry, Single exit (5.3.3)	Same (5.3.4)	Same (20.2.3.(6))	
5. Traceability Upon Interrupt, Save off values	Values of parameters, indices, & other local variables should be saved (5.3.4)	Not included	Not included	
6. Self-Modification	Prohibited (5.3.5)	Same (5.3.5)	Same (20.2.3.(3))	
7. Recursive Modules	Not used, except for stack oriented archi- tecture (5.3.6)	Not included	Not included	
8. Branching (Not Those Done to Simulate Control Structures)	If approved, must only branch forward not back (5.3.8)	Same (5.3.7)	Not included	

PREVIOUS PAGE  
IS BLANK

PREVIOUS PAGE  
IS BLANK

STANDARD:		MIL-STD-1679 Dec 1978	DOD-STD-1679A 22 Oct 83	DOD-STD-SDS (Proposed) 5 Dec 83
9. Indentation/ Nesting	Should be indented for readability (5.3.10)	Same (5.4.10)	Same (5.4.10)	Paragraphing, blocking & indenting (20.2.7) nesting ≤ 3 (20.2.8)
10. Symbolic Parameters	For constants effected by the environment (5.4.1)	Same (5.4.1)		Used for constants, relative locations in a table, & size of data structure (2.2.4)
11. Naming Conventions	Uniform throughout, identify hierarchy position, module, etc., names unique & meaningful (5.4.2)	Same (5.4.2)		Same (20.2.5)
12. Mixed Mode	Avoided when possible, described in comments (5.4.3.2)	Require specific waiver to include (5.4.5)		Same as 1679 (20.2.6)
13. Abstracts	a) description of function b) I/O c) algorithm d) modules called e) called-from f) data description g) data ranges of values h) history i) programmer names	Same except includes: j) company name (5.4.8)		a) purpose b) function, rqmts, interface c) "units" called & calling sequence d) I/O e) global & local variable f) programming dept. g) date of creation h) history (20.2.10)
14. Source Statements	Shall not be compound or complex (5.4.5.3)	Same (5.4.16)		Single statements, non- compound (20.2.9)

STANDARD:	MIL-STD-1679 Dec 1978	DOD-STD-1679A 22 Oct 83	DOD-STD-SDS (Proposed) 5 Dec 83
15. Program Structure & Implementation	Top-down & modular (program constituents) (5.2) Lower level may not call a higher level module	Same	Similar with new software hierarchy terminology
16. Language	HOL only (unless otherwise approved) (5.5.3)	Same (5.3.1)	HOL only (20.2.1) Waiver needed (20.2.2)
Other	Adds Load Maps (5.4.13), cross-reference listings (5.4.12)		

APPENDIX G  
DISTRIBUTION

PREVIOUS PAGE  
IS BLANK

# DISTRIBUTION LIST

<u>Addressee</u>	<u>Number of Copies</u>
Commander U.S. Army Test and Evaluation Command ATTN: AMSTE-TC-M	3
AMSTE-TO	2
AMSTE-EV-S	1
AMSTE-TE	6
Aberdeen Proving Ground, MD 21005-5055	
Commander Defense Technical Information Center ATTN: DTIC-DDR	2
Cameron Station Alexandria, VA 22314-5000	
Commander U.S. Army Aberdeen Proving Ground ATTN: STEAP-MT-M	2
Aberdeen Proving Ground, MD 21005-5000	
Commander U.S. Army Yuma Proving Ground ATTN: STEYP-MSA	2
Yuma, AZ 85634-5000	
Commander U.S. Army Jefferson Proving Ground ATTN: STEJP-TD-E	1
Madison, IN 47250-5000	
Commander U.S. Army Dugway Proving Ground ATTN: STEDP-PO-P	1
Dugway, UT 84022-5000	
Commander U.S. Army Cold Regions Test Center ATTN: STECR-TM	1
APC Seattle, WA 98733-5000	
Commander U.S. Army Electronic Proving Ground ATTN: STEEP-TM-AC	4
Fort Huachuca, AZ 85613-7110	
Commander U.S. Army Tropic Test Center ATTN: STETC-TD-AB	1
APC Miami, FL 34004-5000	

Addressee

Number  
of copies

Commander

U.S. Army White Sands Missile Range

ATTN: STEWS-TE-PY

STEPS-TE-0

STEPS-TE-M

STEPS-TE-A

White Sands Missile Range, NM 88002-5000

4

1

1

1

(BLANK PAGE)

(BLANK PAGE)

**END**

**FILMED**

4-86

**DTIC**